全新的Web GIS开发原理的介绍与应用
详解OGC制定的开放Web服务规范



Web GIS
原理与应用开发

一整套利用开源软件与开放数据开发Web GIS的解决方案

刘光 曾敬文 曾庆丰 著

清华大学出版社



Web GIS
原理与应用开发

刘光 曾敬文 曾庆丰 著

清華大学出版社
北 京

内 容 简 介

互联网与GIS 结合而形成的Web GIS 是GIS 软件发展的必然趋势。本书以循序渐进的方式，通过讲解OGC 制定的相关开放Web 服务规范，介绍了Web GIS 的原理；详解了一整套利用开源软件与开放数据开发Web GIS 的方案，包括空间数据库存储软件PostGIS、数据处理客户端软件QGIS、服务器端软件GeoServer，以及浏览器页面端开发JavaScript API 库OpenLayers，并通过实践的方式，一步一步地介绍这些开源软件的应用，以及如何利用OpenLayers 在互联网上共享地理信息、开发Web GIS2.0 应用；最后，本书还介绍了OpenStreetMap 等开放数据的下载与使用方法。

本书主要读者对象为地理信息系统专业的本科生与硕士研究生，也适用于政府、企业相关部门的GIS 研究与开发人员，还适合作为各种GIS 培训班的学习教材与参考书。

图书在版编目（CIP）数据

Web GIS 原理与应用开发 / 刘光，曾敬文，曾庆丰著. —北京：清华大学出版社，2016
ISBN 978-7-302-44337-7

Ⅰ.①W…　Ⅱ.①刘…　②曾…　③曾…　Ⅲ.①地理信息系统－应用软件　Ⅳ.①P208

中国版本图书馆CIP 数据核字(2016)第166594 号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**17　　**字　　数：**435 千字
版　　次：2016 年9 月第1 版　　**印　　次：**2016 年9 月第1 次印刷
印　　数：1~3000
定　　价：49.00 元

产品编号：066736-01

前　言

计算机科学技术的飞速发展为 GIS（地理信息系统）提供了先进的工具和手段，使 GIS 得到了快速发展，应用日益广泛。进入 21 世纪后，互联网技术的普及更使 GIS 的发展发生了质的变化，互联网已成为 GIS 新的操作平台。互联网与 GIS 的结合即 Web GIS，改变了地理信息的获取、传输、发布、共享和应用的方式。在互联网发布地理信息，为用户提供空间数据浏览、查询、专题图制作与空间分析功能，从而实现地理信息的操作与共享，已经成为 GIS 发展的必然趋势。

Web GIS 概念的首次提出是在 1994 年，指在互联网上部署 GIS，旨在解决冗余数据、昂贵数据的整合以及分布处理能力，将利用新的技术、市场和决策系统来开启我们的世界。Web GIS 是一个分散式的地理信息网络服务，可让地理信息通过 OGC 标准和 W3C 的界面互相沟通存取，凭借良好的互操作性达到以往需要庞大数据量才能实现的功能，使用者可以随意使用在 Web GIS 里的地理空间数据。Web GIS 可让各个符合国际标准的地理信息数据库之间通过 API 方式沟通，从而保证数据不再局限于单一数据库中，可形成网格数据库。Web GIS 是人类社会团体、组织和民众协同合作所建立的信息架构，摆脱以往 GIS 只适用于专业人士的状况，真正地让使用者搜索生活中的各种信息。

早期的 Web GIS 虽然拥有技术上的先进性，但是推广至一般用户较为困难，然而由于近几年 Web 2.0 Mapping 系统的发展，出现了崭新的应用，让以往需要大量数据才能实现的 Web 应用，现在只需要使用 Web 2.0 网站提供的 API 即可实现。Google、Yahoo!、Microsoft 等公司纷纷推出属于自己的地图 API，大大降低了以往开发电子地图的门槛，让许多以 Google Map、Bing Map 等电子地图为显示底图的应用网站如雨后春笋般地发展。例如，有显示性侵害犯罪的 MapSexOffenders.com、反映芝加哥犯罪的 www.chicagocrime.org；结合照片与影像的 Flickr 与 Panoramio；让使用者创造属于自己的地图，并让 Google Map 和其他网页结合的 My Map+；也有提供爱好旅游的使用者通过系统机制和 blog 分享旅游经验，期望建立旅游社群的 MyTripBook；提供飞机航班及时信息的 fboweb.com；结合天气信息的 Weather Underground；租房信息的 housingmaps.com。这些应用都显示了目前电子地图正受到大家的重视，相信未来 Web GIS 2.0 会更加蓬勃发展。

构建 Web GIS 2.0 应用是一个系统工程，包含数据加工处理、数据存储管理、地图制作、地图服务发布、专题数据发布、空间分析功能发布以及系统开发。在该过程中，需要使用多个软件与工具。这些软件与工具既有商业的，同样也有免费与开源的。利用开源软件构建 Web GIS 2.0 应用就是本书介绍的主要内容。

第 1 章在简单回顾 GIS 发展历程之后，着重介绍 Web GIS 的发展以及 Web 服务的重要

性，同时还将介绍自由及开源软件以及在使用过程中存在的优缺点。最后将介绍客户端开源软件 QGIS 的安装与基本使用。

第 2 章介绍 Web GIS 的系统架构与 Web 地图的构成。此外，还介绍用于创建地理 Web 服务的开源软件 GeoServer 的安装与基本应用，包括 GeoServer 的 Web 管理页面及图层预览等。

第 3 章介绍了在自由及开源软件领域存储与处理空间数据的多种选择，列出了空间数据常见的开放格式，各种数据存储结构和格式的优点。最后以实践的方式介绍了如何使用 QGIS 与 GDAL 来处理 GIS 数据，以及如何在 PostGIS 中创建空间数据库并导入空间数据。

第 4 章着重介绍了开放地理空间联盟制定的 WMS 规范。虽然 WMS 并没有使用最新的技术，却是一个被广泛使用的规范，是 Web GIS 的基础。此外，还介绍了如何结合 QGIS 与 GeoServer 发布带高级符号的 WMS 服务。

第 5 章介绍了地图切片的利弊，以及创建与维护地图缓存的策略。最后通过两个实践演示如何在实际工作中创建地图切片。

第 6 章介绍了当前主流的 Web 地图 API，并着重介绍了 OpenLayers 的基本使用方法。

第 7 章介绍了在客户端负责绘制矢量数据的方式与方法。这是当前 Web GIS 的专题图层普遍采用的方式，将所有复杂的符号系统和地图绘制功能转移到客户端，使服务器只需要提供原始的矢量数据和属性数据。这意味着地图引擎可以更有效地响应，从而增强交互性以及提升性能。

第 8 章介绍了引入主流 JavaScript 框架，例如 Dojo、jQuery 等，以便增强 Web 地图的用户体验。此外，还介绍了如何通过专题制图，以更丰富的形式展现空间信息。

第 9 章介绍了 WFS 及其服务的发布、访问与应用，并介绍了如何通过该服务实现基于 Web 的空间数据编辑。

第 10 章介绍了 WCS 服务规范及其在多维数据中的应用，以及如何利用 GeoServer 将带有时间与高程信息的多维数据发布为 WCS 服务。

第 11 章介绍了 WPS 及其服务的发布、访问与应用，并介绍了如何通过 WPS 服务实现基于 Web 的等高线生成以及空间数据的处理。

第 12 章介绍了“开放数据”的不同含义，并介绍了开放数据 OpenStreetMap 及其多种数据下载方法。此外，还介绍了混搭应用及其开发方法。

本书源代码的下载地址为：http://pan.baidu.com/s/1pKSLvVP。如果下载有问题，请电子邮件联系 booksaga@126.com，邮件主题为“求 Web GIS 原理与应用开发源代码”。

本书除了封面署名作者之外，参与本书编写的人员还有刘增良、韩光瞬、唐大仕、刘小东、贺小飞、李珍贵、陈艳玲、杨海、唐伯旺、黄泽清、李凤英、仇诗良和戴海燕等。

由于编者水平、经验有限，书中肯定存在一些疏漏和错误，希望能得到广大读者的批评和指正。

编　者

2016 年 6 月

目　录

第 1 章

Web GIS 概述

从本章可以学习到：

- GIS 的发展
- Web GIS 及其发展
- Web 服务
- 自由及开源软件、开放规范与开放数据
- QGIS 的安装与基本使用

随着计算机、网络和数据库等技术的发展，以及应用的不断深化，地理信息系统（Geographic Information System，GIS）技术的发展呈现出新的特点和趋势，基于互联网的 Web GIS 就是其中之一。Web GIS 除了应用于传统的国土、资源、环境等政府管理领域外，也正在促进与老百姓生活息息相关的车载导航、移动位置服务、智能交通、抢险救灾、城市设施管理、现代物流和大数据分析等产业的迅速发展。

本章在简单回顾 GIS 发展历程之后，将着重介绍 Web GIS 的历史以及 Web 服务的重要性，同时还将介绍自由及开源软件（Free and Open Source Software，FOSS），以及在使用过程中存在的优缺点。最后将介绍一个客户端开源软件 QGIS 的安装与基本使用。本书的后续内容在创建地图服务与 Web GIS 应用时，需要大量使用地理空间数据，因此需要使用 QGIS 来预览并操作这些空间数据。

1.1 GIS 的发展

从一定意义上讲，地理信息系统是计算机和信息系统技术在地理科学中运用发展的产物。因此地理信息系统不仅受地理信息系统的应用和需求的推动，同时也受计算机和信息科学技术的推动。

20 世纪 60 年代末，世界上第一个地理信息系统——加拿大地理信息系统（CGIS）诞生，该系统主要用于自然资源的管理和规划；随后，美国哈佛大学研制出 SYMAP 系统。地理信息系统因日益引起各国政府和科学家的高度重视而迅速发展。GIS 的发展经历了 20 世纪 70 年代的大量试验开发阶段，20 世纪 80 年代的商业开发和运作阶段以及 20 世纪 90 年代以用户为主导的阶段。在 GIS 发展初期，只有地理研究人员、地质调查局、土地森林管理部门、人口调查等专业部门和研究人员对其感兴趣，而目前 GIS 已深入到政府管理、城市规划、科学研究、资源开发利用、测绘和军事等广大的领域。在 21 世纪，地理信息系统已远远不是地理学界或测绘学领域的概念，而将成为人们采集、管理、分析空间数据，共享全球信息资源，为政府管理提供决策、科学研究和实施可持续发展战略的工具和手段。其内涵从狭义的地理信息系统（管理地理信息的计算机系统）到更广泛的空间信息系统（Spatial Information System），并逐渐形成地球信息科学（GeoInformatics）。

从 20 世纪 60 年代以来，计算模式的发展已经经历了从单机计算、集中计算到 C/S 模式、B/S 模式（三层结构模式）的不同阶段，现在正处于以 Web 服务（Web Services）为主要特征的面向服务的计算模式。

就技术层面而言，地理信息系统的发展也经历了四代，如图 1.1 所示。从 GIS 中引入的网络技术方面来看，其中第一代（20 世纪 60 年代～80 年代中期）是以单机单用户为平台、以系统为中心；第二代（20 世纪 80 年代中期～90 年代中期）开始引用网络，实现了多机多用户的 GIS；第三代（20 世纪 90 年代中期～21 世纪初）引入了互联网技术，开始向以数据为中心的方向过渡，实现了较低层次的（浏览型或简单查询型）B/S 结构；第四代（21 世纪初至今）引入了 Web 服务，开始了面向服务的较高层次的 Web GIS。

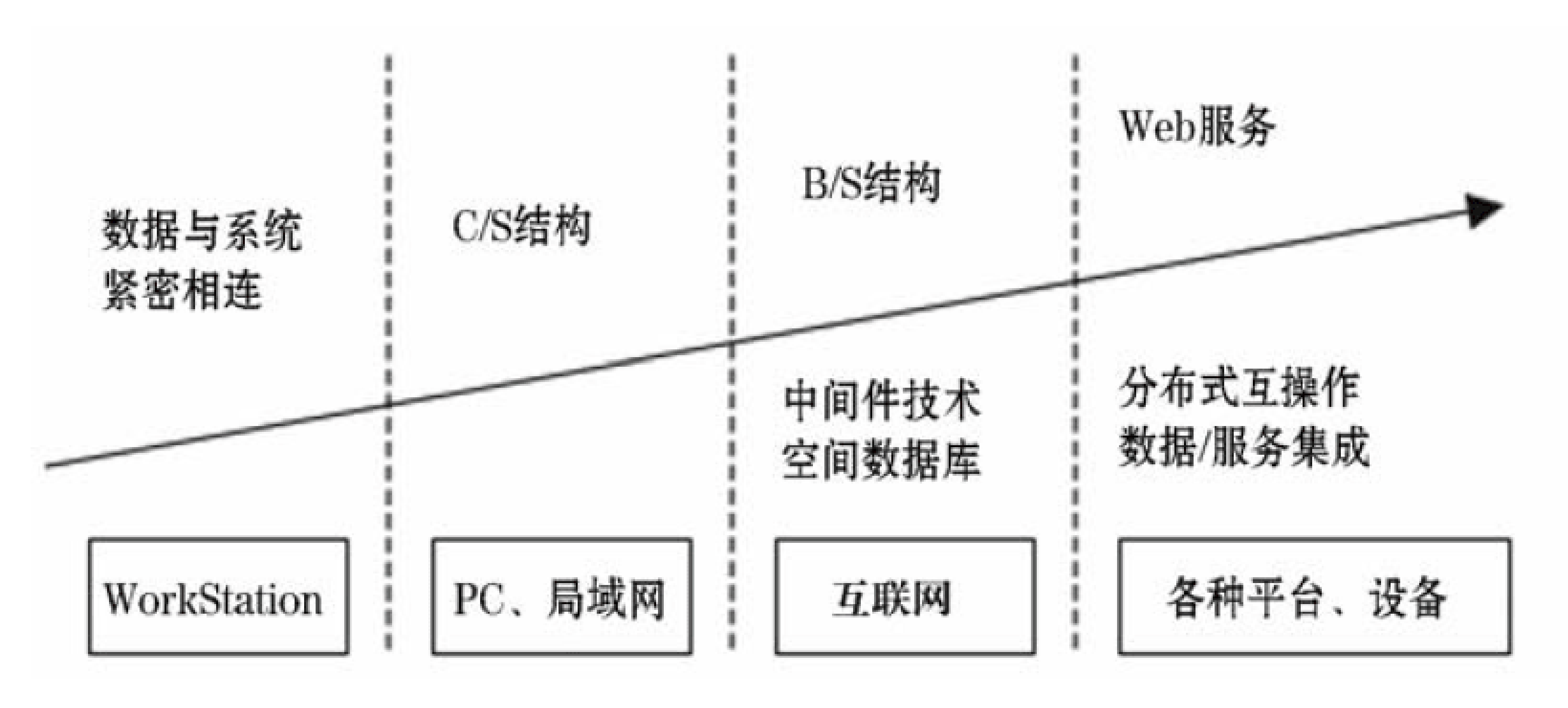


图 1.1　GIS 的发展

1.2　Web GIS 及其发展

在以前的地理信息系统中，基本上以系统为中心，不同系统之间壁垒比较分明，数据共享与服务共享困难。在 30 多年的时间里，形成了许多 GIS 软件，它们在不同的环境中独自发展，有自己的文化背景、领域背景和技术背景，形成了自己的数据模型和功能组织结构。它们在功能和问题描述、实际操作上差别甚大，加上内部空间数据组织不同或者互相保密，形成了不同的壁垒，为信息共享增加了许多困难。

由于互联网技术和 Web 技术的成熟与大规模普及应用，GIS 开始面向传统行业和广大民众，Web GIS 开始出现和发展，并逐渐成为 GIS 应用的一种重要方式。

Web GIS 是将 Web 技术应用于 GIS 开发的产物，是一个交互式的、分布式的、动态的地理信息系统，是由多台主机、多个数据库和无数终端并由客户机与服务器（HTTP 服务器及应用服务器）相连接所组成的。在 Web GIS 中，空间信息应用主要采取的是 B/S（Browser/Server，浏览器/服务器）方式。

Web GIS 的基本思想就是在互联网上提供地理信息服务，让用户通过浏览器从 Web GIS 服务器上获取地理数据和地理处理服务。Web GIS 使全球范围内的用户拥有使用分布式地理信息的能力，用户可以从互联网的任意一个节点，通过 Web 浏览器访问或共享由一个或多个 Web GIS 服务器发布的数据和功能，而不必购买商业 GIS 软件。

Web GIS 的真正意义在于，它将 GIS 从专业应用推向了大众化服务，同时为地理信息共享提供了方便而有效的途径。

1.2.1　传统 Web GIS 的不足

网络技术及分布式计算技术给 GIS 提供了更好的支持，同时也提出了更高的要求。随着网络信息基础设施和技术的不断发展与完善，分布式地理信息服务正成为人们获取地理信息的主要手段。与传统方式相比，分布式地理信息服务具有更广泛的访问范围、平台独立性、低系统成本、更简单的操作等优点，是当前 GIS 发展的重要方向。

但是，在 2005 年之前，传统的 Web GIS（称为 Web GIS 1.0）还有相当多的不足，主要有如下几点：

（1）Web GIS 的主要功能和应用是用于地图的发布，这类系统基本上是浏览型或功能相对简单的查询型系统。虽然有少量对空间数据的操纵，但这种操纵的功能很弱，无法进行复杂的一体化操作，离全面的互操作及分布式的地理信息系统的要求还很遥远。

（2）Web GIS 中主要是服务器端与客户端的通信，由于服务器端与客户端的地位没有形成对等的实体，因而难以建立分布式的地理信息系统。

（3）Web GIS 中传递的数据主要是以矢量形式表达的少量地图数据或者是以栅格形式表达的地图，这样的地图数据在各个应用系统中的格式不统一，语义也不统一。由于缺乏统一的标准，数据的共享难以实现。

（4）Web GIS 中实现的操作，在各个系统中没有统一描述的机制（虽然也有一些系统制定了一定的查询语言，如 GeoSQL，但这不是所有的系统都采用的），也没有对这些操作和服务提供注册和发现的机制，因此服务的共享难以实现。

（5）Web GIS 还没有形成一套有效的集成机制。新一代的 GIS 要求具有有效的分布式空间、数据管理和计算，包括：多用户同步空间数据操作与处理机制；数据、服务代理和多级 B/S 体系结构；异构 GIS 系统互连与互操作；空间数据分布式存储与数据安全；空间数据高效压缩与解压缩；同时要求强大的应用集成能力，包括有效的遥感、地理信息系统、全球定位系统集成；强大的应用模型支持能力；GIS 与 MIS（管理信息系统），特别是 ERP（企业资源计划）的有机集成；GIS 与 OA（办公自动化）的有机集成；GIS 与 CAD（计算机辅助设计）的有机集成；GIS 与 DCS（决策支持系统）的有机集成；有一定实时能力、微型化、嵌入式 GIS 与各类设备的集成等。

综上所述，GIS 中大量的数据不断积累、各种层次的软件也越来越多，Web 技术的发展给 GIS 提出了更高的要求，GIS 的分布式、可互操作性显得越来越重要，这恰恰是当前 Web GIS 要着重解决的问题，这也是新一代（即第四代）GIS 的一个重要发展方向。

这个时期主要的商业 Web GIS 产品有 MapInfo 公司的 MapXtreme 产品系列、ESRI 公司的 IMS 产品系列以及 Intergraph 公司的 Geomedia Web Map 产品等，我国也推出了 Geo-Surf、GeoBeans 与 SupperMap IS 等国产化的 Web GIS 产品。而大型的面向公众的 Web GIS 门户网站则非常少，在国外主要以 MapQuest 为代表，在国内主要有图形天下（Go2Map）和城市通（ChinaQuest）。

1.2.2 从 Web 站点发展为 Web 服务

随着 Web 技术、组件技术、分布式系统等技术的发展，在 21 世纪初出现了 Web 服务技术，并逐渐引起了人们的注意，并成为分布式异构 GIS 进行互操作集成的首选技术。

在 Web 应用的不断发展过程中，人们发现在 Web 应用和传统桌面应用（比如企业内部管理系统、办公自动化系统等）之间存在着连接的鸿沟，人们不得不重复地将数据在 Web 应用和传统桌面应用之间转换，这成为了阻碍 Web 应用进入主流工作流的一个巨大障碍。

从 1998 年开始发展的 XML 技术及其相关技术已证明可以解决这个问题，而随后蓬勃发展的 Web 服务技术则正是基于 XML 技术的针对这一问题的最佳（在当时看来）解决方案。Web 服务的主要目标就是在现有的各种异构平台的基础上构筑一个通用的、与平台和语言无关的技术层，各种不同平台之上的应用依靠这个技术层来实施彼此的连接和集成。Web 服务与传统 Web 应用技术的差异在于：传统 Web 应用技术解决的问题是如何让人来使用 Web 应用所提供的服务，而 Web 服务则要解决如何让计算机系统来使用 Web 应用所提供的服务。

将 Web 服务应用于 GIS，则可以使传统的地理信息系统由独立的 C/S 结构或 B/S 结构，实现到基于 Web 服务体系的 GIS 的跨越。

从 OGC（Open Geospatial Consortium，开放地理空间信息联盟，在 1994~2004 年该机构名为 Open GIS Consortium）制定的规范名称中也可以看出向 Web 服务的发展趋势，从 OGC01-065：Web Feature Server Implementation Specification 到 OGC02-058：Web Feature Service Implementation Specification（如图 1.2 所示），原先用 Server，后来用 Service，这实际上体现了从传统的 Web GIS 向 Web 服务观念的转变。

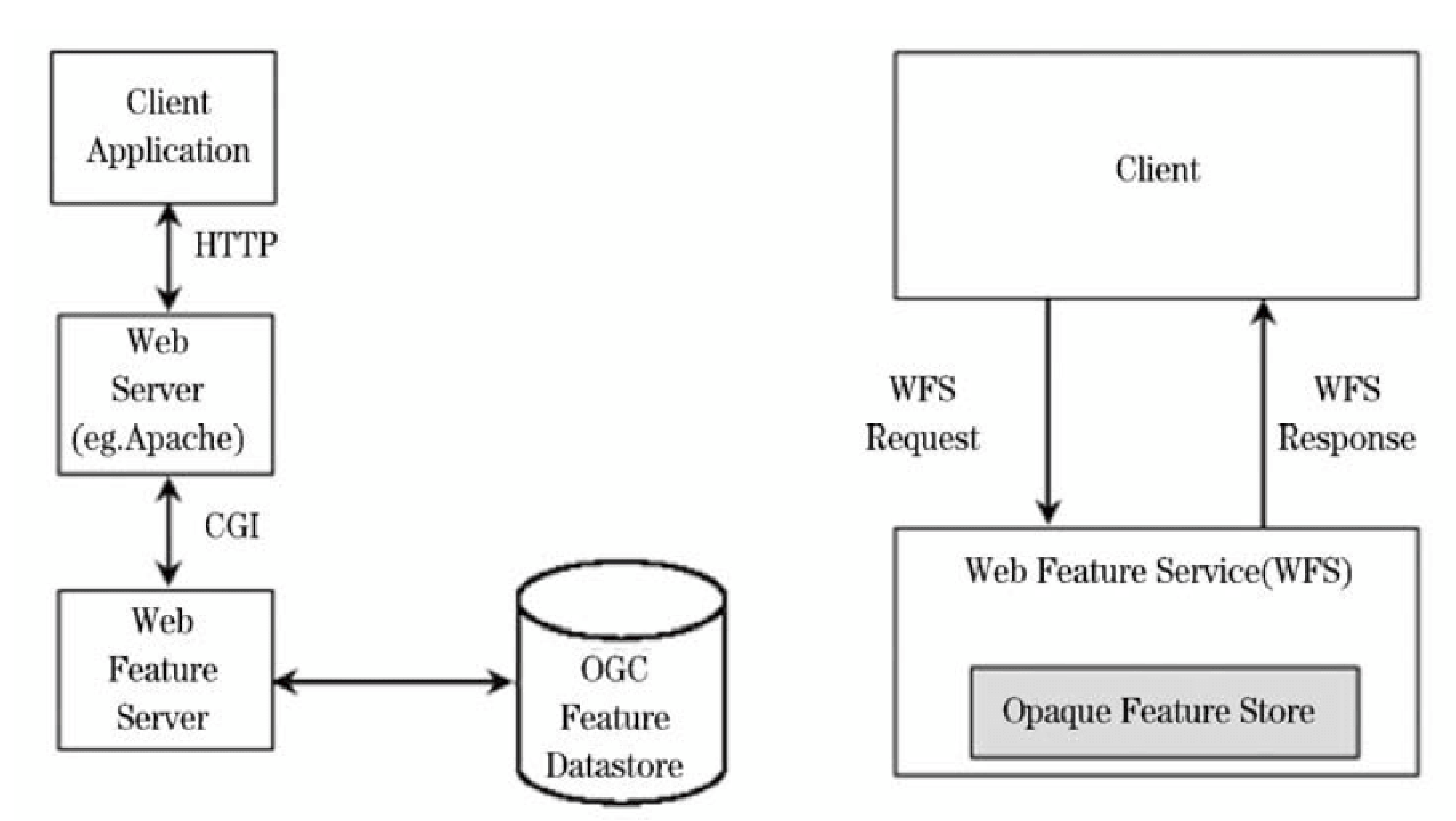


图 1.2　传统 Web GIS 与基于 Web 服务的 GIS 的对比

1.2.3　从 SOAP 发展为 REST

在 Web 服务发展的初期，XML 格式化消息的第一个主要用途是应用于 XML-RPC 协议，其中 RPC（Remote Procedure Call，远程过程调用）代表远程过程调用。在 XML 远程过程调用（XML-RPC）中，客户端发送一条特定消息，该消息中必须包括名称、运行服务的程序以及输入参数。

之后为了标准化，跨平台又产生了基于 SOAP（Simple Object Access Protocol，简单对象访问协议）的消息通信模型。SOAP 是在 XML-RPC 基础上，使用标准的 XML 描述了 RPC 的请求信息（URI/类/方法/参数/返回值）。因为 XML-RPC 只能使用有限的数据类型种类和一些简单的数据结构，SOAP 能支持更多的类型和数据结构。优点是跨语言，非常适合异步

通信和针对松耦合的 C/S，缺点是必须在运行时做很多检查。

随着时间的推移和 SOAP 的推广情况，大家很快发现其实世界上已经存在了一个最为开放、最为通用的应用协议，那就是 HTTP，使用 SOAP 的确让进程间通信变得简单易用，但并不是每个厂商都愿意将自己的老系统再升级为支持 SOAP，而且 SOAP 的解析也并不是每种语言都内置支持，比如 JavaScript。而 HTTP 正好完美解决了这个问题，因此可以设计一种使用 HTTP 协议来完成服务端与客户端通信的方法，于是 REST（Representational State Transfer，表达性状态转移）应运而生。REST 一般用来和 SOAP 作比较，它采用简单的 URL 方式来代替一个对象，优点是轻量、可读性较好且不需要其他类库支持，缺点是 URL 可能会很长且不容易解析。

1.2.4　从三层架构发展为多层架构

早期的 Web GIS 通常是典型的三层的 B/S 架构，如图 1.3 所示，包括前端的 Web 浏览器，后台 Web 服务器以及数据服务器，而 GIS 服务功能则内嵌在 Web 应用程序中。

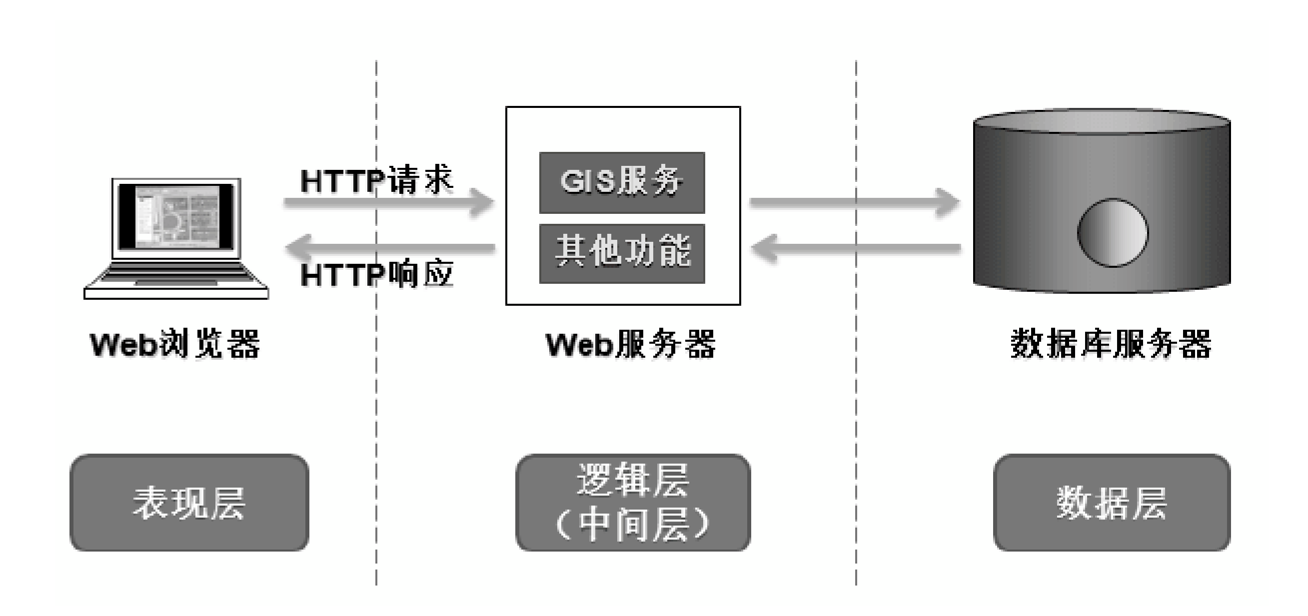


图 1.3　典型的三层 Web GIS 架构图

但是为了在更大程度上方便地理信息数据及 GIS 功能的共享，以及方便二次开发，通常将 GIS 服务单独部署，这时的系统架构如图 1.4 所示。

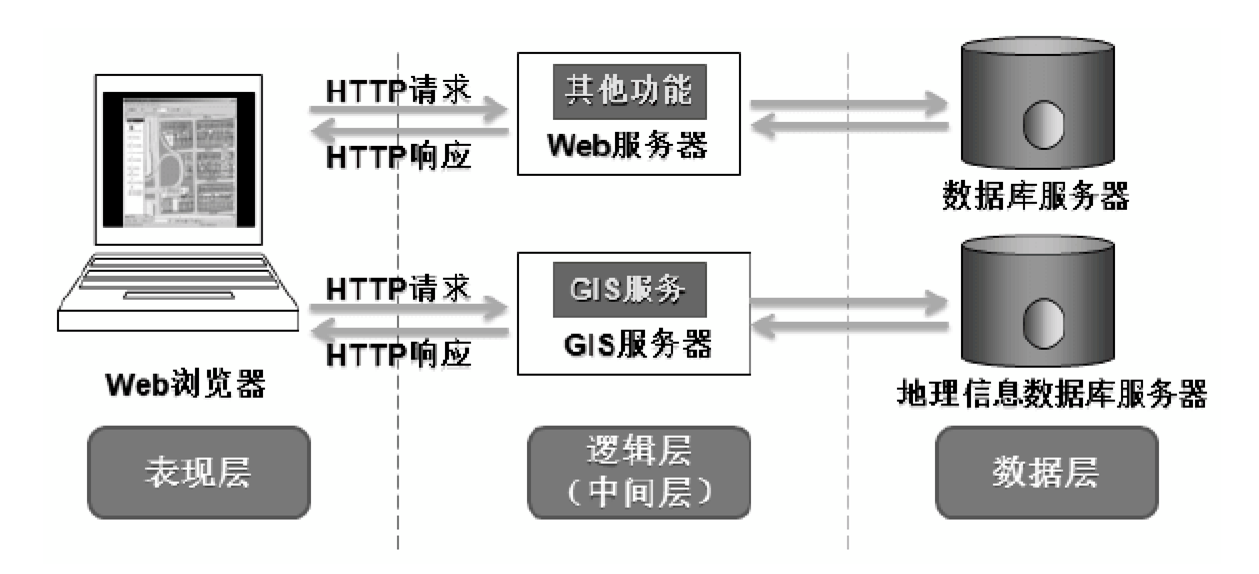


图 1.4　四层架构的 Web GIS

在上述系统架构中，可以利用现有的 GIS 服务，例如 Google、Microsoft、百度、高德的地图服务，也可以利用商业 ArcGIS Server 或开源的 GeoServer 等地理信息服务软件，将地理信息发布为服务，在系统客户端利用 JavaScript 调用这些服务，从而在系统中集成地图及 GIS 功能。

正因为 Web 服务的独立性以及 Ajax 等交互技术，Web GIS 的浏览器端程序可以访问多个 Web 服务资源，而不仅仅是本系统拥有者所开发的 Web 服务，这就演化为如图 1.5 所示的混搭架构。

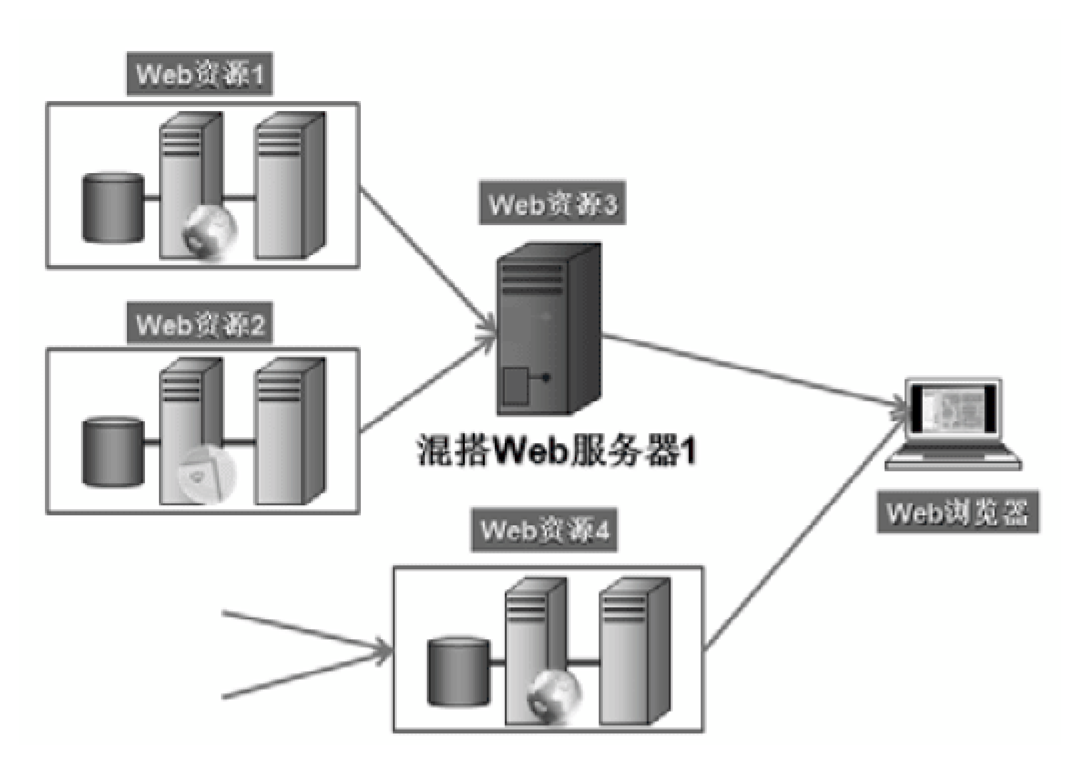


图 1.5　多层架构的 Web GIS

混搭技术指整合网络上多个资料来源或功能，以创造新服务的网络应用程序。该词源自于流行音乐将两种不同风格的音乐混合，以产生新的趣味的作法。虽然在古老的 HTML 2.0 版本中早有这个概念（将图片提供视为一种服务，一个网页中的文字与图片可以来自不同的网站，一个图文并茂的网页就是一种原始的混搭），一般还是将混搭技术视为 Web 2.0 的特性之一。Web 技术的这种发展为 GIS 的实施提供了一种新的模式。一个用户可以从一个服务器获得一层信息，再从另一个服务中获取其他数据或专业模型，将它们融合在一起，进而产生基于 Web 的新的 GIS 应用模式。这种新的模式将极大地拓展 GIS 的应用范畴和服务领域。

1.2.5　从 Web GIS 1.0 到 2.0

早期的 Web GIS 是依据当时的网络环境提出的，近年来由于 Web 2.0（主要包括 Web 服务、REST 与 AJAX 等技术）的迅速发展，原本 Web GIS 中所依赖的方法与技术也不断在更新，表 1.1 显示了 Web GIS 1.0 与 Web GIS 2.0 之间的一些重要区别。

表 1.1　Web GIS 1.0 与 Web GIS 2.0 之间的重要区别

Web GIS 1.0	Web GIS 2.0
静态的二维地图	动态的二维全球性地图，使用者互动性高（例如 Google 地图、必应地图等）
文件传输（FTP）	直接使用网络服务
地理数据交换中心	地理网络服务目录入口网站（例如 Geodata.gov 等）
独立 Web 站点	网络服务融入技术
使用者端点服务	远程网络服务 API（例如 Google Map API、ArcGIS API for JavaScript、OpenLayer 等）
数据单方向给予	使用者参与地理数据制作（例如 Open Street Map）
使用者发表意见困难	互动机制提升，使用者之间交流增加（例如 Blog 等）
私人通信协议	标准的通信协议（例如 OGC 标准、SOAP、WSDL、REST 等）

Web GIS 1.0 主要关注的是静态二维地图，Web GIS 2.0 主要关注二维动态地图和对三维地图的研究（例如 Google 地图、Microsoft 必应地图和 ESRI ArcGIS Explorer）。这些 Web GIS 2.0 新增的技术提升了用户体验，而且让使用地理网络技术的用户拓展了一个数量级。Web GIS 获取地理信息的方式同时也发生了转变，从使用 FTP（文件传输协议）来传输地理信息方式，转变为直接使用数据流的 Web 服务和一组 API。另一个重要变换是使用混搭技术。

1.3 Web 服务

上一节中简单介绍了 Web 服务，由于 Web 服务是当前 Web GIS 核心内容，因此有必要更深入地详细介绍。

1.3.1 Web 服务的重要性

1.3.1.1 从数据共享的角度看空间信息 Web 服务

空间数据共享一直是 GIS 发展的瓶颈。地理空间数据是研究地球形成演化、探讨人类生存环境、减轻自然灾害、合理开发资源和促进社会可持续发展的重要科学依据。随着网络技术及 Web GIS 的飞速发展和应用，更加迫切要求社会各部门能够共享地理空间数据以及与之相关的资料，实现地理空间信息的全球共享。地理空间信息共享的重要性主要体现在：

- 为决策全球化问题同时提供大量的科学地理数据。
- 地理信息系统（GIS）发展的强烈要求。
- 为使用空间信息的社会各部门节省大量的人才、时间和金钱。
- 便于实现空间信息的规范化和标准化。
- 可以加强空间信息的高效管理、维护、重复利用和有效增值。
- 为数字地球的建设奠定基础。

空间信息共享是指使得查询、浏览、获取、交换、使用和再加工地球上与人类生存直接或间接相关的信息能够做到方便、快捷、准确、安全和全面，包括对部分信息处理资源的自由使用。空间信息共享强调空间数据集之间的相互透明访问和信息用户对数据集的透明访问与使用，注重从空间信息的语义层次、数据模型层次和数据结构层次消除空间信息描述方法上的差异性以及表示方法上的差异性，对空间信息给出统一的描述和表示，达到空间信息本质上和形式上的共享。

空间信息共享活动涉及了 3 个主要概念：空间信息资源、空间信息的获取与处理和空间信息应用。而这样也就出现了 3 种不同的角色：空间信息提供者、空间信息处理软件和提供者——就是通常所谓的 GIS 以及用户。

空间信息共享要解决可达性、互操作性和易用性。

可达性是指用户能存取到数据。通过空间信息 Web 服务，用户可以通过其中提供的数据服务，来查找、获取用户感兴趣的数据，这种数据可以分布于网络上，并由不同的服务商来提供不同区域、不同专题、不同质量的数据。

互操作性是指在不同的 GIS 之间能互相操作、对数据能进行相同的理解。GIS 互操作的关键就是解决空间信息异构问题。而信息具有语法和语义，可以分层次讨论信息异构问题。因此在互联网环境中的空间信息共享，通过空间信息 Web 服务，可以进行语法及语义差别的消除工作。在消除空间信息资源的语法差异方面，通过 Web Services 可以在数据的请求者与服务者之间用统一的数据格式，这种数据格式包括 GML 在内，它们已逐渐形成标准并在 Web 服务中使用。在消除空间信息资源的语义差异方面，语义 Web 等方面的进展也可以对此有所帮助。

在数据共享所采用的技术方面，包括数据格式转换，通过 API 直接读取，是比较低层的方式；基于 DBMS 的集成，则可以使数据更统一，具有一定的互操作性；在基于 Web Services 进行集成的系统中，数据有统一的 Schema 描述，数据之间在进行接口时有统一的协议和标准，可以有效地实现数据的共享，如图 1.6 所示。

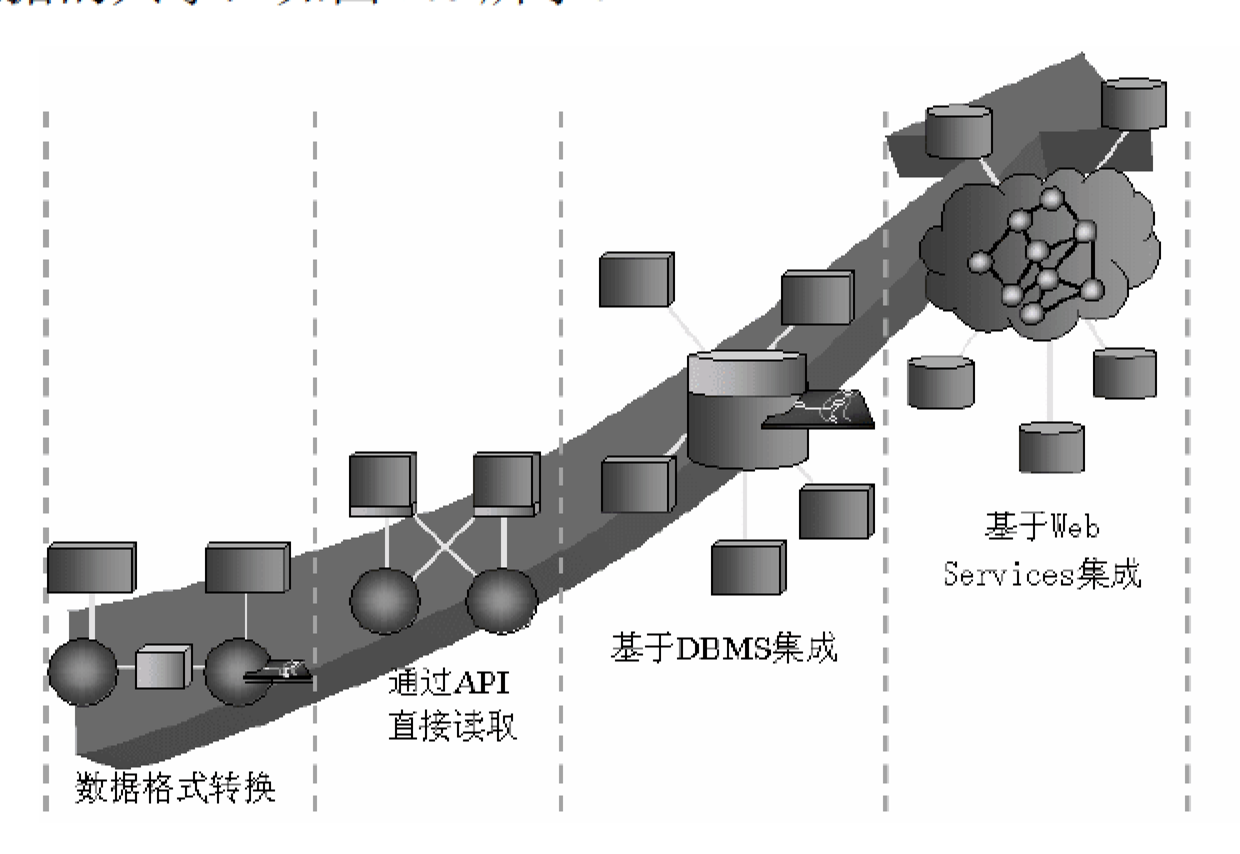


图 1.6　数据共享方式

1.3.1.2　从软件复用的角度看空间信息 Web 服务

GIS 技术经过长期的发展，特别是近年来的长足进步，已经积累了大量的基础软件及应用软件，这些软件的功能不同、编程方法不同、编程接口不同且用户界面也不相同。在新的网络日益发展、软件技术飞速进步的今天，如何有效地发挥这些软件的功能，充分利用已有的软件或软件构件，来实现功能更强的软件或快速实现一个应用系统，也是一个重要问题。存在大量的可复用构件是有效复用的基本前提，而有效地管理大量构件，提供方便的构件存储、构件检索和构件提取等功能，则是成功复用的必要保证。Web 服务系统则为这种复用和管理提供了一种技术支持。

首先，将现有的 GIS 系统中的功能改造成 Web 服务是可行的。由于 Web 服务采用的技术基础是 XML，它是一种规范化的文本，易于被各种编程语言进行处理，这使得可以利用现有的 GIS 系统中的功能进行分解、重组、规范化，从而提供 Web 服务接口是可以实施的。

其次，在 Web 服务中的对象实际是网络上分布的对象。这些对象不论其内部如何实现，但在这些对象之间是靠通用的接口来进行通信的，这些接口遵守各种层次的协议，如 XML、SOAP、WSDL 等。对于空间信息 Web 服务中的服务，还有与空间信息处理相关的协议，如 WFS（Web Feature Service，网络要素服务）、WCS（Web Coverage Service，网络地理覆盖服务）等。一个系统可以方便地调用远程对象，而不论这种对象是在哪种平台上，也不论这种对象是用什么编程技术来实现的，这样就通过 Web 服务的对象复用实现了更高层次的对象复用。

另外，Web 服务是服务的松散集合，可以方便地发布，并可以通过程序自动或人工地进行查找和调用，这就给利用已有的 Web 服务带来了方便。例如，可以将现有的多个不同供应商提供的地图显示服务集合起来，再形成一个新的、面向专题的地图服务。

1.3.1.3　从系统集成的角度看空间信息 Web 服务

在一定意义上 Web 服务技术是一种系统集成的技术。Web 服务用于 GIS 是一种更好的进行 GIS 系统集成的技术。

GIS 系统的集成技术经历了以下的发展过程。

1. 基于对象技术的 RPC 方法

多年以来，应用程序分布式集成计算方法的演化基本上都是基于远端过程调用（RPC，Remote Procedure Call）机制。RPC 是指应用程序对运行在远端计算机上的代码进行功能调用。为实现该调用的请求，在相互协作的计算机之间，需要使用特定的协议支持来对信息数据进行打包、发送和接收。

目前基本上每种主要的对象技术均有其自己的 RPC 技术。Microsoft 组件对象模型（COM）使用 DCOM/COM+，CORBA 使用 IIOP，Java 使用 RMI。这种以“分布式对象”为标志的集成分布式系统，在应用于集成更广泛的分布式系统时具有很大的不足。由于这些应用程序接口需要“严格匹配”，并与目标系统的专用技术密切相关，所以其连接非常脆弱，很难实现真正的跨平台、异构系统的集成。由于这些相互关联的组件可能各不相同，所以开发和维护的费用非常高。由于在等待远端所访问资源的响应时，对组件进行的同步调用经常“阻塞”，所以其可扩展性也存在固有的缺陷。

2. 基于消息中间件技术的 RPC 方法

基于消息的中间件技术引入了关联的方法，以便使用消息传送技术集成更大地域范围中的分布式系统，改进了对象集成技术的可扩展性和可管理性。由于这样通常不会阻塞应用程序的调用，应用程序可以“发送并忘记”信息，从而使操作更有弹性和可扩展性。Microsoft、IBM、BEA 等公司都提供消息队列与事务处理中间件产品，支持分布式应用系统的集成。

基于“消息中间件”的集成技术的一个主要限制就是它的开放性。首先，即使系统最初可以接受开放数据格式，它们仍然基于专用程序很高的技术，并使用专用的接口和专用的内部数据表示方法；其次，购买这些技术、集成和后续维护昂贵，且这些技术通常需要在应用

程序连接的两端同时运行特定的软件或各有一份客户许可协议，这样增加了成本、消耗时间并增加了集成的复杂性，集成的范围也受到限制。因此需要一种开放的系统集成技术，使用符合工业标准的传输协议、过程调用和数据表示方法，以支持平台分布式系统的低成本集成与互操作。

3. SOAP 风格的 Web 服务方法

早期 Web 服务方法起源于 XML-RPC 技术。在 1998 年，Userland、DevelopMentor 与 Microsoft 一起发布了 XML-RPC（http://www.xmlrpc.com）。XML-RPC 提供了一种简单的机制，使处于不同环境下的应用之间，可以通过互联网来进行远程过程调用。它采用 XML 作为编码标准，HTTP 为传输协议。XML-RPC 通过 HTTP 请求向 RPC 服务器提交请求，通过 HTTP 响应 Response 获得 RPC 的调用结果。这种 XML-RPC 技术突出地强调了 XML 和 HTTP 的作用，使用应用的集成可以跨平台地进行。后来，IBM、Microsoft、SAP 等公司共同在此基础上逐步提出了 Web Services 的概念，并制定了一系列服务调用、发布、集成的协议，使 Web Services 成为新一代的系统集成方法。

SOAP 风格的 Web 服务将应用逻辑封装起来，对于用户只提供标准接口。对于客户端，服务器上的数据和应用逻辑是透明的，因此它能够灵活组织网络资源，较好地解决地图的共享问题。但是基于 SOAP 的 Web 服务依赖于定制，每个 SOAP 消息使用独特的命名资源的方法，每个 SOAP 应用需要定义自己的接口，SOAP 的这些特点对于服务间的互操作的实现十分不利；另外，SOAP 协议栈并不是专门为 GIS 而设计，没有考虑 GIS 数据具有空间参考、海量存储等特点，除了简单、小规模的 GIS 应用，SOAP 协议栈很难不加改动地应用在地理信息服务领域。

4. REST 风格的 Web 服务方法

REST 为解决上述问题提供了新的契机，它以更贴近 WWW 基础协议的方式来实现 Web 服务，大大简化了 Web 服务的设计与调用。因此当前更流行的是基于 REST 风格的 Web 服务。

空间信息 Web 服务就是将这种新的技术应用于地理信息系统，解决地理信息系统中的多源、异构、分布系统的集成问题。

1.3.2　REST 及 REST 风格的 Web 服务

因为 REST 风格的 Web 服务与复杂的 SOAP 和 XML-RPC 相比而言明显更加简洁，越来越多的 Web 服务开始采用 REST 风格设计和实现。例如，Amazon.com 提供 REST 风格的 Web 服务进行图书查找；雅虎提供的 Web 服务也是 REST 风格的；国内百度地图、高德地图提供的地图服务也是 REST 风格的。REST 对于资源型服务接口来说很合适，同时特别适合对于效率要求很高、而安全要求不高的场景。本书后面内容中发布与使用的也是 REST 风格的 Web 服务，因此这里再详细介绍其特点。

1.3.2.1 REST

单纯就 REST 术语的出现而言，它是由 Roy Fielding 在 2000 年的论文中首次提出的一种软件架构。

REST 基础概念包括：

（1）在 REST 中的一切都被认为是一种资源。每个资源由 URI 标识。

（2）对资源的操作包括获取、创建、修改和删除资源，这些操作正好对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。也就是说使用统一的接口，而不像 SOAP 风格的服务那样，每个服务的名称都是不同的。

（3）无状态。每个请求是一个独立的请求。从客户端到服务器的每个请求都必须包含所有必要的信息，以便于理解。

（4）资源的表现形式则是 JSON（JavaScript Object Notation）、XML 或者 HTML，取决于读者是机器还是人，是消费 Web 服务的客户软件还是 Web 浏览器。当然也可以是任何其他的格式。

REST 架构风格最重要的约束有如下 6 个方面：

- 客户/服务器

通信只能由客户端单方面发起，表现为请求/响应的形式。

- 无状态

通信的会话状态应该全部由客户端负责维护。

- 缓存

响应内容可以在通信链的某处被缓存，以改善网络效率。

- 统一接口

在通信链的组件之间通过统一的接口相互通信，以提高交互的可见性。

- 分层系统

通过限制组件的行为（即每个组件只能“看到”与其交互的紧邻层），将架构分解为若干等级的层。

- 按需代码（可选）

支持通过下载并执行一些代码（例如 Java Applet、Flash 或 JavaScript），对客户端的功能进行扩展。

1.3.2.2　REST 风格的 Web 服务

REST 风格的 Web 服务(也称为 REST 风格的 Web API)是一个使用 HTTP 并遵循 REST 原则的 Web 服务。它从以下 3 个方面资源进行定义：

- URI，比如：http://example.com/resources/。
- Web 服务接受与返回的互联网媒体类型，比如：JSON、XML、YAML 等。
- Web 服务在该资源上所支持的一系列请求方法(例如 POST、GET、PUT 或 DELETE)。

表 1.2 列出了在实现 REST 风格 Web 服务时 HTTP 请求方法的典型用途。

表 1.2　HTTP 请求方法在 REST 风格 Web 服务中的典型应用

资源	GET	PUT	POST	DELETE
一组资源的 URI，比如：http://example.com/resources/	列出 URI，以及该资源组中每个资源的详细信息（后者可选）	使用给定的一组资源替换当前整组资源	在本组资源中创建/追加一个新的资源。该操作往往返回新资源的 URL	删除整组资源
单个资源的 URI，比如：http://example.com/resources/142	获取指定资源的详细信息，可以自选一个合适的网络媒体类型（比如 XML、JSON 等）	替换/创建指定的资源。并将其追加到相应的资源组中	把指定的资源当作一个资源组，并在其下创建/追加一个新的元素，使其隶属于当前资源	删除指定的元素

1.3.3　查看在线的 Web 服务

那么如何访问这些 REST 风格的空间信息 Web 服务呢？一般包括以下 4 个步骤。

（1）构建请求 URL。

URL 由服务的端点加参数构成。

首先确定端点。每个 GIS 服务都有一个端点，例如一个典型的 WMS（Web Map Services，网络地图服务）地图服务的端点为：

http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi?SERVICE=WMS

然后确定操作。不同 GIS Web 服务支持不同的操作。不同的操作会返回不同的结果。地图服务可以输出地图、识别某要素。输出地图可以生成地图，同时识别出地图服务层的属性表。例如 WMS 提供 GetMap 操作，返回指定范围的地图图片。

接着确定参数。不同的操作需要不同的参数。例如，如果请求地图图片，需要提供地图范围的四周角点坐标参数，也就是地图覆盖范围。

最后确定输出格式。

例如一个访问美国雷达地图的 URL 格式为：

http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi?SERVICE=WMS&REQUEST=G

etMap&FORMAT=image/png&TRANSPARENT=TRUE&STYLES=&VERSION=1.3.0&LAYERS=nexrad-n0r&WIDTH=877&HEIGHT=276&CRS=EPSG:900913&BBOX=-15252263.28954773,2902486.4758432545,-6671748.242369267,5602853.811101243

（2）提交 URL 请求到服务器。可以不通过编程发送 URL 请求。例如，只需要在网页浏览器的地址栏中输入网址。每种编程语言都有不同的提出请求方式。

（3）服务器处理请求，并运行 Web 服务代码，然后将结果返回到客户端。该结果通常是一张图片或包含信息的字符串。

（4）解析和使用响应。

并不是所有的请求都调用 Web 服务代码，一些 Web 请求内容仅仅返回一个文件。而这就是切片地图工作原理，也就是切片地图服务快的原因。在后面的内容中将更详细介绍切片地图。现在来观察一个指定缩放级别、行与列的切片的请求：

http://a.tile.openstreetmap.org/15/11068/19742.png

该请求下钻到服务器的文件结构中，并将请求的 PNG 图片作为响应返回给客户端。在服务器端除了检索文件之外，并没有运行代码，因此也可以说并没有调用 Web 服务。但是，这类简单的 Web 请求同样是许多 Web GIS 的重要组成部分。

并不是所有的 Web 服务都是用同样格式的 URL 与参数。本书介绍了 Web 服务最常用的格式，特别是那些开放标准与开源软件的 Web 服务。

虽然我们在访问 Google 地图、必应地图以及百度地图等网站时，并不需要使用上述复杂 URL，但是浏览器确实是在发送成百上千这样的请求到服务器上。下面通过 Firebug 这一 Mozilla Firefox 浏览器的插件，介绍查看浏览器在后台如何发送这些请求的方法。

（1）单击 Firefox 浏览器上的“工具”选项，然后单击“附加软件”，在弹出的小窗口中，单击右下角的“获取扩展”选项，在打开的页面中搜索 Firebug，最后在搜索结果的页面中下载 Firebug 并安装。

（2）重新启动 Firefox 浏览器，打开网址为 http://map.baidu.com/的百度地图网站。

（3）单击状态栏中最右方的灰色昆虫按钮（），或使用快捷键 F12 启用 Firebug 插件，它会将当前页面分成上下两个框架，并且此时灰色按钮变成红色（）。如果有两个显示器，最好使用快捷键 Ctrl+F12，在另一个窗口打开 Firebug。

（4）在 Firebug 窗口中，切换到“网络”面板。如果是第一次使用，则需要使用“网络”标签旁的下拉箭头启用监视网络功能。

（5）在百度地图窗口中进行漫游、放大缩小等操作。

（6）这时在“网络”面板中会显示发送的许多 Web 请求，大部分是请求地图切片。将鼠标悬停到某个请求上，如果请求返回的不是图片，则会显示完整的 URL，否则还会显示响应图片的缩略图，如图 1.7 所示。

（7）展开请求，可以查看请求更为详细的细节，包括发送的参数、头信息、响应以及缓存等。

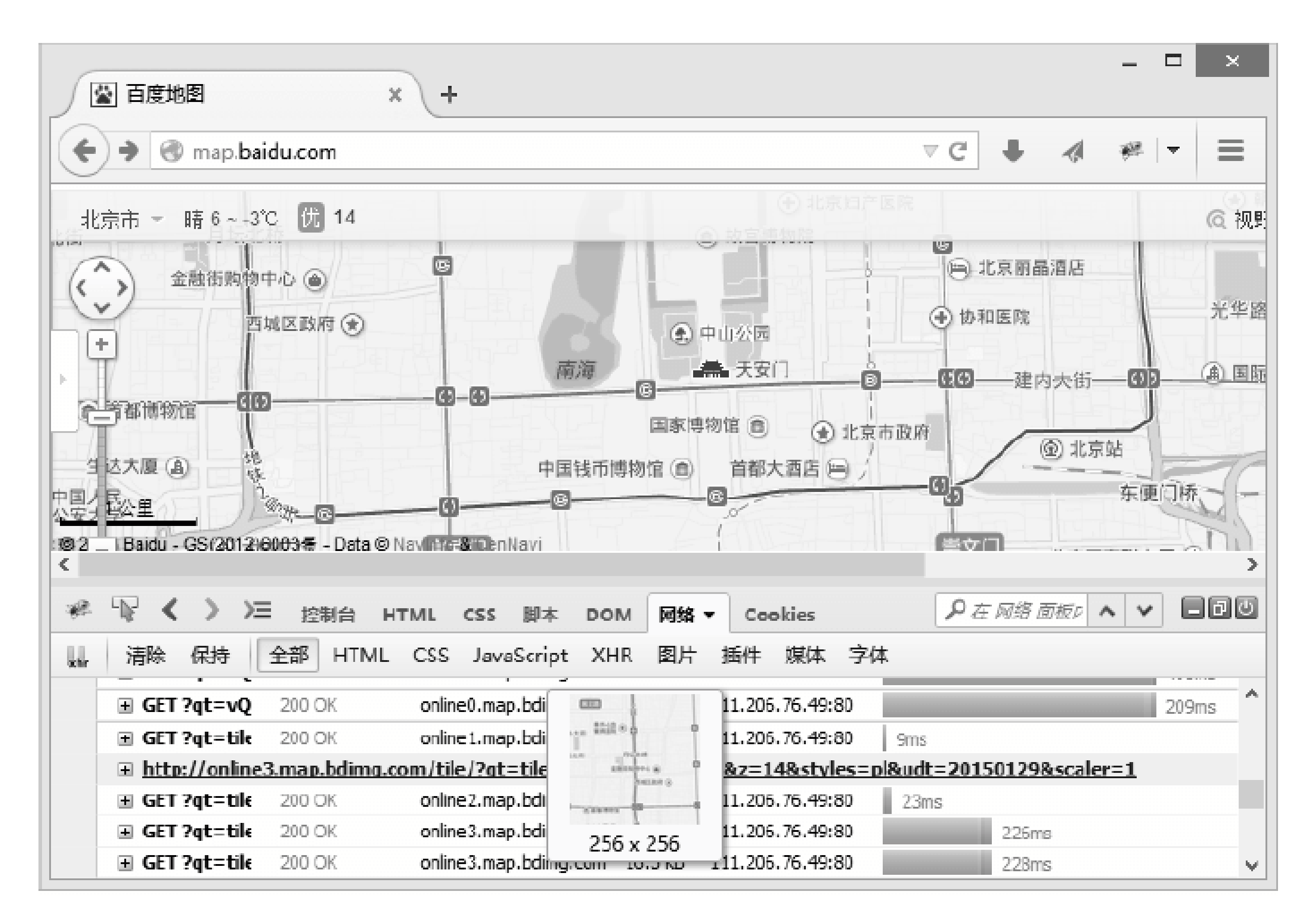


图 1.7　利用 Firebug 查看浏览器发送的请求

Firebug 是 Mozilla Firefox 浏览器的开源扩展，提供了很多工具，可以监视、编辑和调试任何 Web 站点的级联样式表（CSS）、HTML、文档对象模型（DOM）和 JavaScript。Firebug 包括一个 JavaScript 控制台、一个日志记录 API 以及一有用的网络监视器。利用 Firebug 可从各个不同的角度剖析 Web 页面内部的细节层面，可以很轻松地调试和优化 Web 和 Ajax 应用程序，给 Web 开发者带来很大的便利。

其他最新的浏览器一般也都包含类似的“开发人员工具”（Internet Explorer）、“开发者工具”（Google Chrome）菜单。

1.3.4　OGC 的 Web 服务规范

在 GIS 分布式、互操作方面，一些组织（如 OGC，UCGIS 等）正在制定相应标准和进行一些实验项目，其中 OGC 在空间信息 Web 服务方面制定了一系列规范，统称为 OWS。

在 OWS 服务体系中，主要包括如下 6 个部分，如图 1.8 所示。

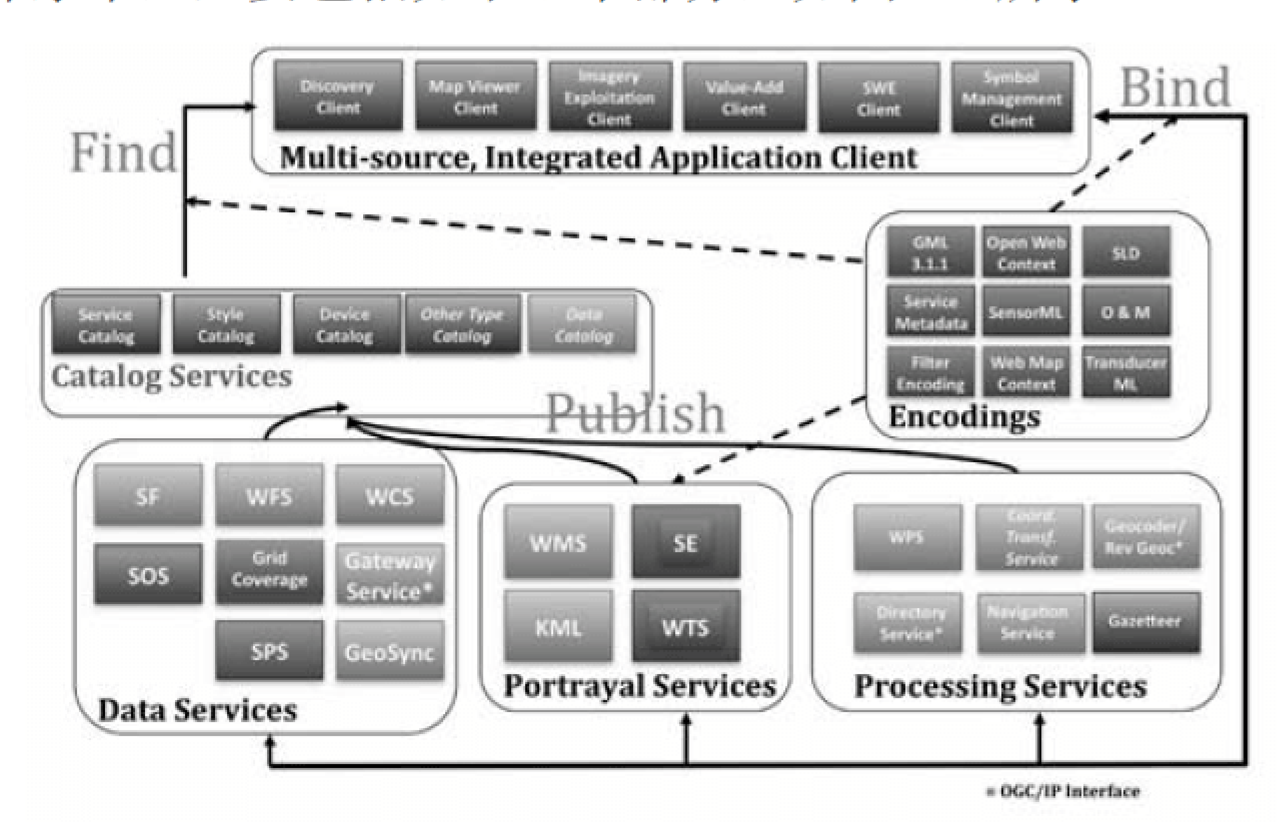


图 1.8　OWS 的组成

（1）地理数据服务（Data Service）。

提供对空间数据的服务，主要有 WFS、WCS。地理数据服务返回的结果通常是带有空间参照系的数据。

（2）描绘服务（Portrayal Service）。

提供对地理空间信息进行可视化的能力。给定一个或多个输入，描绘服务会产生渲染后的输出，例如地图的制图描绘、地形的透视图、影像的注解图、特征地物在时间和空间上的动态改变等，主要有 WMS。

（3）处理服务（Processing Service）。

提供对地理空间数据进行操作的服务和增值服务，主要有 WPS（Web Processing Service，网络处理服务）、Geocoder（地理编码服务）、Gazetteer（地名辞典服务）、Coordinate Transfer Service（坐标转换服务）等。

（4）目录服务（Catalog Services）。

提供对各种服务、数据及相关资源的描述信息（即元数据）集合的发布与查询，包括服务目录、样式目录、数据目录、设备目录以及其他类型的目录。

（5）编码（Encodings）。

所有 OGC 框架的编码规范都采用 XML Schema 来定义。这些编码描述了特定的词汇表，用于在应用客户和服务之间、服务与服务之间封装成消息的数据传输。OGC 的规范定义或计划定义的编码主要包括 GML（Geography Markup Language，地理置标语言）、SLD（Styled Layer Descriptor，样式化图层描述符）以及 Service Metadata（服务元数据）等。

（6）客户端应用（Client Application）。

即客户端的基本应用，如地图的显示、地图浏览以及其他一些增值服务。

1.4 自由及开源软件、开放规范与开放数据

当前商业 GIS 软件的使用和维护费用越来越高，例如包含客户端与服务器端一整套的 ESRI ArcGIS 软件售价约为 70 万元人民币。而且其销售策略是，若购买了服务器端软件则必须购买客户端软件，其理由是用户既然使用了其服务器端软件来发布服务，那必然就需要使用其客户端软件来处理数据。这对一些比较小的 Web GIS 应用来说，远远超出了其可承受的范围。并且众多商业软件 GIS 的数据和操作并非完全能够转换和共享，造成一些信息孤岛。不过在商业 GIS 软件的对面活跃着开源 GIS。OGC 成立于 1994 年，致力于研究和建立开放地理数据交互操作标准，使用户和开发者能进行互操作。国际地理空间开发基金会（Open Source Geospatial Foundation）成立于 2006 年 2 月，其使命是支持开源地理信息软件和遥感软件的开发及推动其更广泛的应用，并对其支持的项目提供组织、法律和财政上的帮助，促进 OSGeo 基金会基于地理信息开发标准软件及其互操作技术的开发、推广和普及。OSGeo 中国中心于 2006 年 9 月成立，帮助中国地区的用户和开发者更好地使用 OSGeo 基金会提供的源代码、产品和服务。

1.4.1 自由及开源 GIS 软件

自由及开源软件是一种可以归类为既是自由软件又是开源软件的计算机软件。也就是说，任何人被授权后都可以自由地使用、复制、研究和以任何方式来改动软件，并且其源代码是开放和共享的，因此人们被鼓励志愿地改善软件的设计。这种软件是相对于商业软件而言，而后者是在版权的严格限制之下，并且通常其源代码对于用户是不开放的。

当前存在许多包含了各种层次的自由及开源的 GIS 软件，例如大型的桌面 GIS 有 QGIS 和 GRASS GIS 等，目前比较流行的服务器端软件有 Geoserver、MapServer 和 QGIS Server 等，还有开源的 GIS 数据库项目如 PostGIS/PostgreSQL Spatial Database，另外还有一些数据转换工具（如 OGR 和 GDAL）以及地图投影算法库（如 Proj4 和 Geotrans）等开源项目。这些软件大多都得到 OSGeo 的支持。

GRASS 是大型的 GIS 系统，最早由美国军方建筑工程研究实验室构建维护，后来贡献给开源社区，目前 GRASS 已经覆盖大多数 GIS 系统的操作函数，有超过 300 个经典算法。GRASS 就是一个开源版本的 ArcGIS。QGIS 是一个用户界面友好的桌面 GIS 系统，使用基于 QT 的图形库实现，大名鼎鼎的 KDE 图形和 Google Earth 也是基于 QT 构建。而且 QGIS 可以很好地支持 GRASS 的算法接口，成为了 GRASS 的一个重要的前端表现工具，QGIS 就是一个简化版本的 GRASS。GRASS 和 QGIS 都是跨平台的桌面 GIS 软件，可以运行在很多操作系统上。本书将介绍在将数据发布为 Web 服务之前如何使用 QGIS 浏览并操作数据。

MapServer 底层采用 C 语言来编写，基于 CGI 脚本实现，页面调用支持 PHP、JSP、JavaScript 等多种语言，并对 OGC 的 WMS 和 WFS 提供支持。GeoServer 基于 Java 和 GeoTools 库开发，它的功能全面遵循 OGC 开放标准，GeoServer 对发布 WFS-T 和 WMS 提供便捷的支持，并以 XML 文件描述所有地图服务。本书将介绍如何用 GeoServer 来发布空间信息 Web 服务。

PostgreSQL 是一个开源的基于对象的数据，PostGIS 则是为 PostgreSQL 提供空间支持，类似 ArcGIS 的空间数据引擎 ArcSDE。本书也会介绍如何使用 PostGIS 来存储空间数据。

对于需要使用 API 将地图图层组合起来并显示在网页中，其中最为成熟的是 OpenLayers。OpenLayers 是一个用户开发 Web GIS 客户端的 JavaScript 包，实现访问地理空间数据的方法都符合行业标准，采用面向对象方式开发，并使用来自 Prototype.js 和 Rico 中的一些组件。本书将会介绍如何使用 OpenLayer 来访问空间信息 Web 服务。其他对应的 FOSS API 包括 Leaflet、ModestMaps、D3 以及 Polymaps 等。

1.4.2 开放规范的使用

这里要介绍的是两类开放规范。

（1）开放数据格式。

如果某数据格式有完备的描述文档，对于各种 GIS 软件都能读写，并且格式的发明者没有宣称任何版税的权利，那么该数据格式就是开放的。

文本格式的开放数据格式包括 KML（Keyhole Markup Language，Keyhole 标记语言）、

GeoJSON 与 TopoJSON 等，JPEG 与 PNG 是栅格中开放的数据格式。ESRI Shapefile（shp）简称为 Shapefile，是美国环境系统研究所公司（ESRI）开发的一种空间数据开放格式。目前，该文件格式已经成为了地理信息软件界的一个开放标准，这表明 ESRI 公司在全球的地理信息系统市场的重要性。Shapefile 也是一种重要的交换格式，它能够在 ESRI 与其他公司的产品之间进行数据互操作。与此相反的是 ESRI 的文件地理数据库（File Geodatabase），该格式是封闭的，除了 ESRI 的工具之外，其他软件不能打开与创建。

（2）开放规范。

GIS 界最主要的开放规范就是 OGC 制定的一系列规范（已经在“1.3.4 OGC 的 Web 服务规范”中进行了介绍）。国际标准化组织（ISO）技术委员会 211（TC211）也是最主要的空间信息标准组织之一。目前 ISO/TC 211 已经完成或正在制定的地理信息国际标准约有 40 余项，包括《地理信息参考模型》《地理信息概念模式语言》与《地理信息术语》等。

此外，ESRI 与多个其他组织合作进行开发，正在推动开放式 GeoServices REST 规范的使用。此规范为 Web 客户端利用 REST 技术与 GIS 服务器进行通信提供了标准方法。通过 ArcGIS for Server 发布的 Web 服务遵守此规范。这意味着非 ESRI 的开发者可以自由地创建应用来发布与访问符合该标准的 Web 服务。虽然 GeoServices REST 并没有得到 OGC 的采纳，但这是一个商业软件自愿公开规范的典型例子。

1.4.3 开放数据的作用

开放数据是一类可以被任何人免费使用、再利用、再分发的数据。在其限制上，最多是要求署名和使用类似的协议再分发。

Data.gov 中包含了许多由美国政府收集的开放数据。此外，开放街道地图（OpenStreetMap，缩写 OSM）也是一个广泛使用的开放数据源，如图 1.9 所示。OSM 项目由英国人 Steve Coast 创立，概念启发自维基百科网站，是一个构建自由内容之网上地图协作计划，目标是创造一个内容自由且能让所有人编辑的世界地图，并且让廉价的移动设备有方便的导航方案。本书将在后面的内容介绍如何从中获取数据。



图 1.9　北京西站附近开放街道地图数据

1.5　实践 1：QGIS 的安装与基本使用

在将数据发布为 Web 服务之前，需要进行收集数据、格式转换、数据编辑、样式设置等这些大量的数据准备工作，这些准备工作是创建 Web GIS 的一部分。这些工作需要使用客户端软件，本书介绍使用的是 QGIS 这一 FOSS 软件。

在该实践中，将介绍如何安装 QGIS 及其矢量数据的处理操作。

（1）下载 Shapefile 数据。

请从前言中给出的下载地址下载本书提供的资源。这里所使用的数据位于“Data\Ottawa”目录中，是加拿大安大略省渥太华市的数据。

（2）下载 QGIS 安装程序。

读者可以访问 QGIS 官方网站（www.qgis.org），在其主页单击“Download Now”按钮，进入下载页面。对于 Windows 操作系统的用户，可根据操作系统是 32 位还是 64 位，选择 QGIS Standalone Installer Version 2.6（32 bit）或 QGIS Standalone Installer Version 2.6（64 bit）。本书编写时最高版本是 2.6，如果读者在阅读时 QGIS 推出了更高版本的，可以下载高版本的。

对于 32 位的 Windows 操作系统的读者，也可使用本书提供的安装程序，即位于 Tools 文件夹中的 QGIS-OSGeo4W-2.6.1-1-Setup-x86.exe 文件。

（3）安装 QGIS。

安装时一切都选择默认设置即可。安装完成后，桌面上不仅包含了 QGIS Desktop 程序快捷键，此外还包含了其他 4 个应用程序的快捷键，包括 GRASS GIS、OSGeo4W 等。

（4）添加矢量文件。

打开 QGIS Desktop 程序。若要加入矢量文件，可以通过窗口左边的“浏览器”定位到要加入的文件，然后将其拖到“图层”管理器中即可。也可使用按钮，打开“添加矢量文件”对话框，然后通过“浏览”按钮找到需要的文件。不过要说明的是，虽然一个 Shapefile 同时包含几个文件，但是只需要选择文件扩展名为.shp 的即可。按照上述两种方法之一打开 roads.shp 文件。

（5）设置符号。

在“图层”管理器中双击 roads 图层，就会弹出“图层属性”对话框，在左边部分选择“样式”标签，切换到“样式”面板中。由于 roads 图层包含的是线要素，因此在该面板中可以设置线的颜色、线的宽度、可视比例范围以及注记等。通过“样式”面板，将 roads 图层的线符号设置为细灰线，即 Residential road 符号。

（6）设置注记。

在“图层属性”对话框中的左边部分选择“标签”标签，切换到“标签”面板。首先将注记字段设置为 name，并将注记文本颜色设置为灰色。然后切换到“位置”小面板，将注记与道路线之间的距离设置为 5 毫米，如图 1.10 所示。

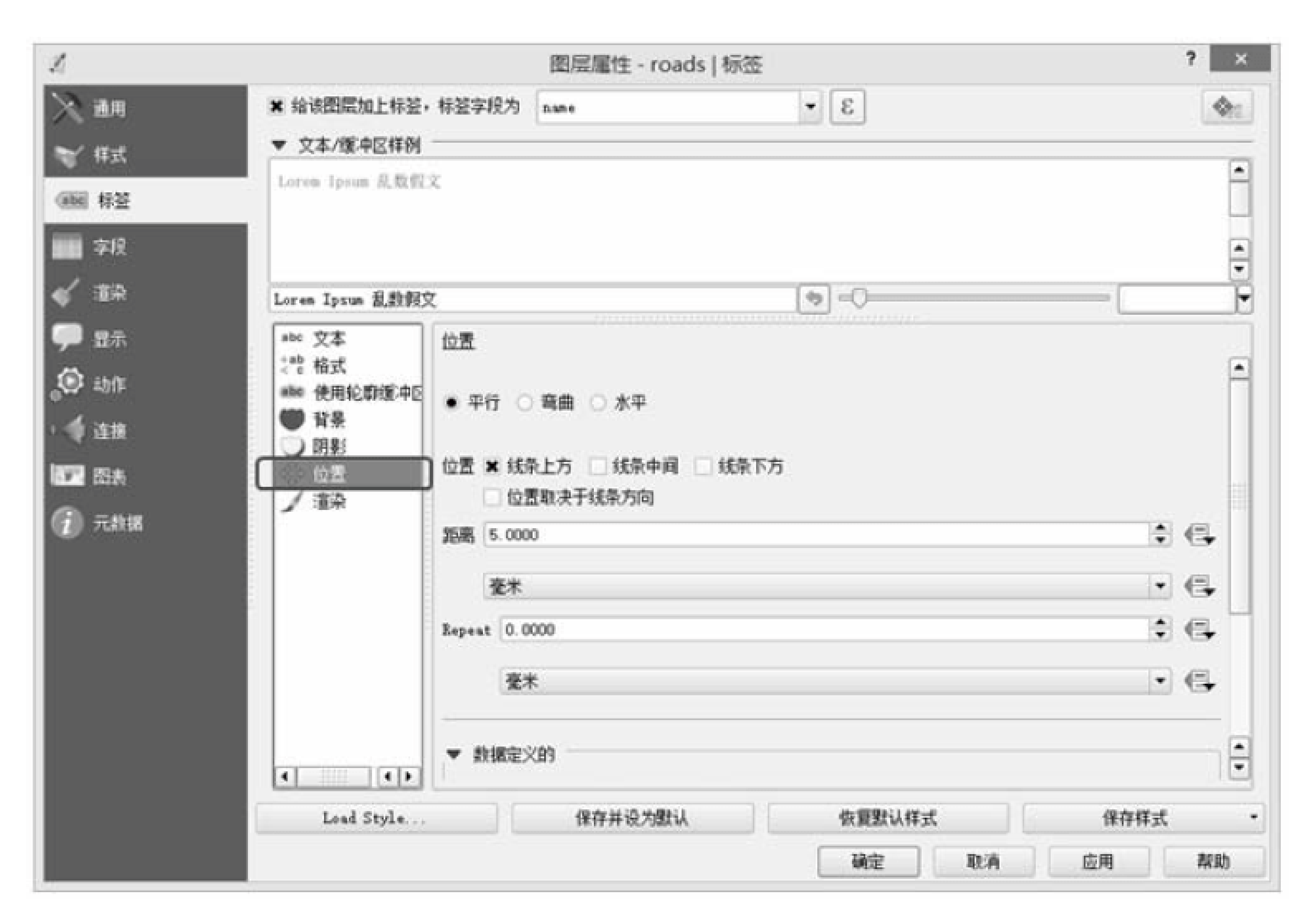


图 1.10　设置图层注记的位置

在“标签”面板中选择“渲染”标签，在其中将注记显示的最小比例尺设置为 1:10000，以避免在小于该比例尺时还显示注记。为了防止重复注记，可在“要素选项”部分选择“合并相连的线条以防止重复注记” 选项。

通过这样的设置后，可得到如图 1.11 所示的效果。

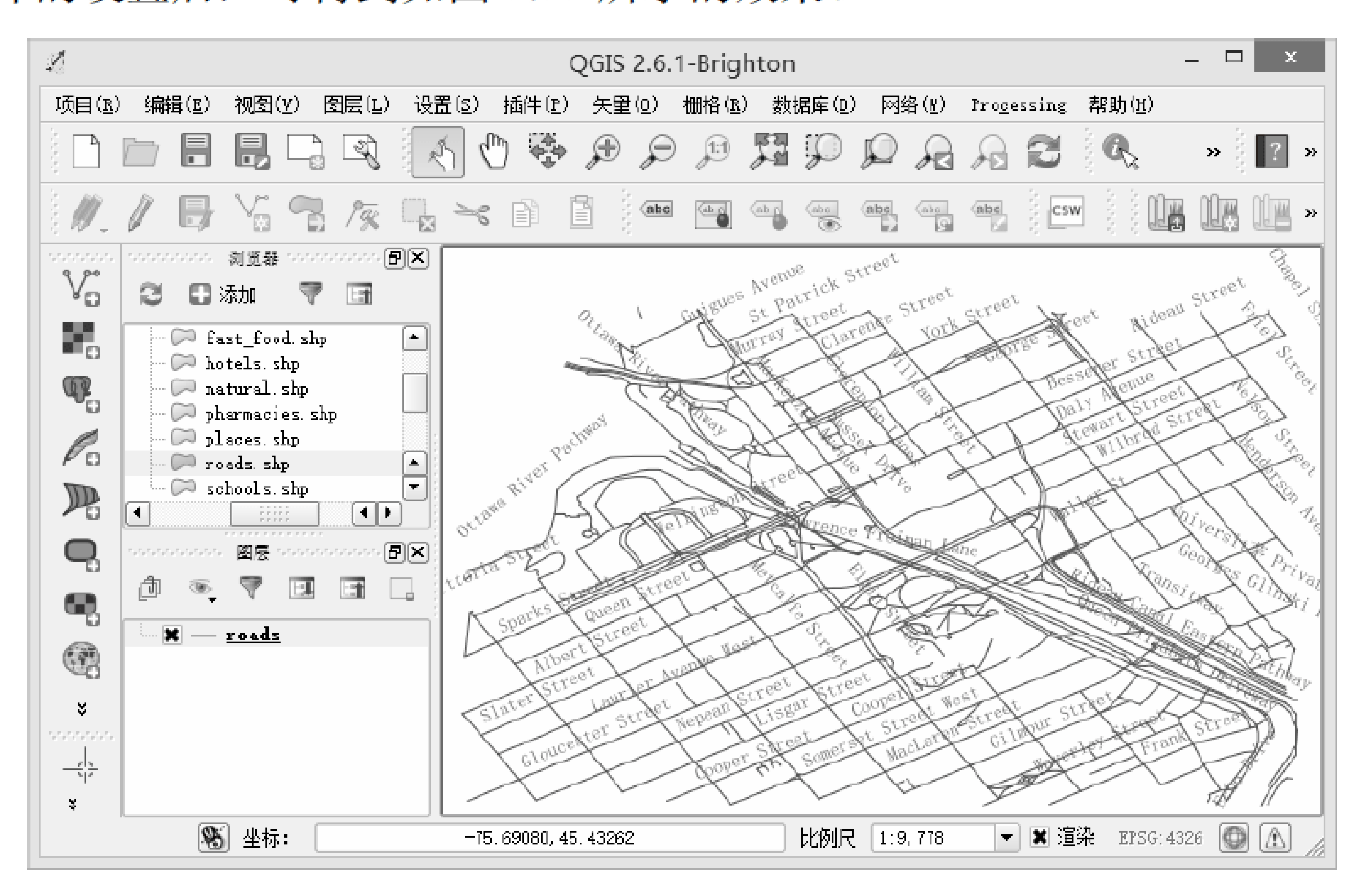


图 1.11　设置道路的符号与注记之后的效果

（7）设置 natural 图层。

在地图中加入 natural.shp 数据。将其样式设置为不带边界的淡绿色填充符号。需要按照图 1.12 的方式，将“边界样式”设置为“不显示画笔”才能实现该目的。



图 1.12　设置不显示边界的符号

（8）设置 buildings 图层。

在地图中加入 buildings.shp 数据，这是建筑物图层。对于建筑物，当然最好是让其有突出的效果。而这可以使用多个符号来实现，步骤如图 1.13 所示。



图 1.13　对同一图层设置多个符号

（9）在地图中加入 bus_stops.shp，这是公交站点的数据。

将其样式设置为蓝色填充的公交 SVG 标记。SVG 指可伸缩矢量图形（Scalable Vector Graphics），使用 XML 格式定义图形。SVG 图像在放大或改变尺寸的情况下其图形质量不会有所损失。并在“图层属性”对话框的“通用”面板中，将其显示的最小比例尺设置为 1:10000。在“通用”面板中还可以设置图层中“图层”管理器中显示的名称。例如 bus_stops.shp 默认显示为 bus_stops，我们可以将其设置显示为“公交站点”，更直观易懂。

（10）设置地图提示。

在图层管理器中高亮选择“公交站点”图层，然后单击“地图提示工具”按钮，这时如果将鼠标悬停在某公交站点要素上时，就会显示该站点的名称。

配置完成后，整个地图显示效果如图 1.14 所示。



图 1.14　设置多个图层样式后的地图

（11）保存地图。

选择“项目”菜单中的“保存”菜单项，将地图保存为 Ottawa.qgs。

QGIS 使用.qgs 保存地图，为 XML 格式，可以使用文本编辑器打开。

1.6　习　题

（1）利用 Firebug 工具查看 Google 地图（ditu.google.cn）、必应地图（cn.bing.com/ditu/）、开放街道地图（www.openstreetmap.org）以及天地图（www.tianditu.cn）网站。

- 了解这些网站是如何组织地图切片的，有什么异同。
- 进一步了解 Firebug 的使用。

（2）在 Ottawa.qgs 地图中再加入其他矢量文件，并配置符号与注记等，使地图在各种比例尺下都比较美观而且内容丰富。

（3）用文本编辑器打开 Ottawa.qgs 文件，查看 QGIS 是如何组织地图的。

（4）阅读以下材料，了解商业软件公司是如何参与开源运动的。

- 开源技术与 ESRI（http://www.esri.com/news/arcnews/spring11articles/open-source-technology-and-esri.html）。
- ESRI 与开源（http://www.esri.com/products/arcgis-capabilities/open-source）。

第 2 章

Web 服务与 Web GIS 的设计

从本章可以学习到：

- Web GIS 的系统架构
- Web 地图的组成
- GeoServer 的安装与初步使用

在第 1 章中介绍了 Web GIS 的发展历史。那么，要利用计算机中的原始地理空间数据集，创建一个美观、交互性强、成千上万用户使用的 Web GIS 应用，应该从哪里开始呢？首先必须了解当前 Web GIS 的系统架构，从宏观层面把握 Web GIS 的构成。除了系统架构之外，本章还将介绍 Web 地图的构成，包括基础地图、专题图层、交互小组件以及每个构成部分的角色与作用。最后将介绍用于创建地理 Web 服务的开源软件 GeoServer 的安装与初步应用，包括 GeoServer 的 Web 管理页面及图层预览等。

2.1 Web GIS 的系统架构

通常一个 Web GIS 应用系统会使用几台物理计算机来存储数据、处理数据、制作地图、发布服务以及访问应用等，一般使用系统架构图来描述，这些不同的计算机处于不同的层中。虽然，本书中实践部分为了方便读者操作，使用一台计算机来同时承担这些不同的角色，但是必须要清楚这些不同的层的作用，以及它们是如何协同构成一个完整的系统。

一个完整的 Web GIS 系统架构包括数据服务器、GIS 服务器、Web 服务器与使用该系统的各种终端（客户端、移动端、浏览器等）以及服务管理员与服务发布者，如图 2.1 所示。



图 2.1　Web GIS 系统架构图

（1）数据服务器是很容易理解的，用于存储 GIS 服务所需要的各种空间数据，有可能是文件服务器，存储了地图切片或 Shapefile 等格式的空间数据，也可能是数据库服务器，以数据库的方式存储空间数据。这些服务器通常会要求有冗余存储机制以及定期备份脚本，以防止数据丢失。

（2）GIS 服务器是安装在服务器机器上的核心软件，该软件用于创建 Web 服务。这些 Web 服务包括绘制地图、同步数据库、投影几何对象、搜索数据并执行许多其他空间，分析操作。由于 GIS 服务器是一个 Web GIS 的核心，一般都要求性能较高、处理能力强的，不过可以同时使用多台 GIS 服务器。在这个多个 GIS 服务器中可以根据服务器的性能或者根据应用的不同而进行分组，不同的组用于处理不同的服务，比如说性能比较好的机器用于处理地理处理服务，性能一般的用于处理地图服务，这种结构如图 2.2 所示。

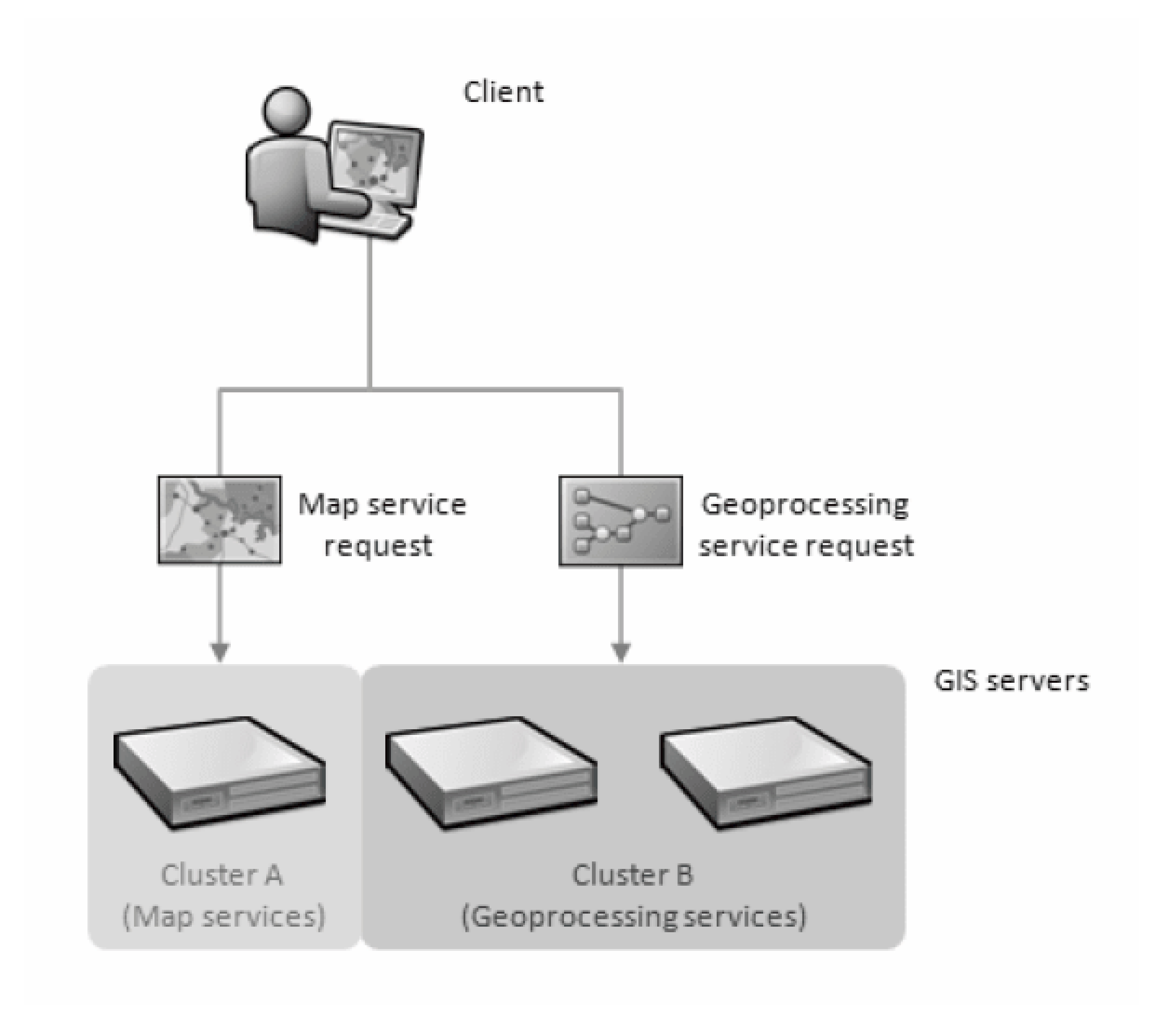


图 2.2　不同性能的机器处理不同类型的服务

本书将使用 GeoServer 来发布空间信息 Web 服务。

（3）Web 服务器是运行 Web GIS 应用程序或非空间信息 Web 服务的计算机。该服务器是访问 Web GIS 的入口，也是放置 Web GIS 应用程序代码（包括 HTML 与 JavaScript 文件等）的地方。

在安装 GeoServer 时，也会同时安装一个内嵌的 Jetty 的 Web 服务器。Jetty 是一个开源的 Servlet 容器，它为基于 Java 的 Web 内容，例如 JSP 和 Servlet 提供运行环境。Jetty 是使用 Java 语言编写的，它的 API 以一组 JAR 包的形式发布。开发人员可以将 Jetty 容器实例化成一个对象，可以迅速为一些独立运行（stand-alone）的 Java 应用提供网络和 Web 连接。因此，为了测试，可以将代码放置在 Jetty 的主文件夹下。本书中后面会介绍该方法。

当然，虽然 GeoServer 是 Java 的应用程序，但是其发布的是 Web 服务，因此对于访问该服务的 Web GIS 应用，可以使用.NET 编写，也可以使用 Java 编写。对于.NET 编写的应用程序，可以部署在 IIS（Internet Information Services，互联网信息服务）Web 服务器上。本书后面也会介绍该方法。

此外，由于当前主流的 Web GIS 直接使用 JavaScript 访问 Web 服务，而 JavaScript 是运行在浏览器上的，因此是直接访问 Web 服务，而不需要 Web 服务器。对于那些没有使用.NET 或 Java 代码的 Web GIS，作为测试，则并不需要将这些代码部署到 Web 服务器中，直接在浏览器中打开包含 JavaScript 代码的 HTML 页面即可。当然，如果需要其他计算机的浏览器也能访问该 Web GIS 应用，那么必须将其部署到 Web 服务器中。本书后面也会介绍该方法。

（4）Web GIS 应用需要管理员来安装软件、配置 Web 应用程序以及调整站点以获取最佳性能。GeoServer 管理员使用 GeoServer Web 管理页面来发布与管理空间信息 Web 服务。管理员可以寻求开发人员的帮助或自己学习脚本技巧，从而通过 GeoServer Administrator API 自动执行管理任务。内容创作者需要使用客户端应用程序来创建要发布到站点的 GIS 资源（例如地图和地理数据库）。在将资源发布到服务器的过程中，这些应用程序也可以起到辅助作

用。本书介绍使用 QGIS 以及 GDAL 等命令行工具来完成这些任务。

（5）Web、移动和桌面应用程序都可连接到 GIS 服务器发布的空间信息 Web 服务。这些应用程序的终端用户依靠 Web 服务来获得 GIS 数据或实现分析，但是它们可能不知道有关该 Web 服务的详细信息或者不知道可获得哪些服务。当规划部署的规模和范围时，全面了解访问 Web 服务的终端用户数以及它们对该站点的使用模式很有价值。

2.2 Web 地图的组成

Web 地图是引用一组地图和空间信息 Web 服务构成的有效地图，可以在任意客户端进行使用（桌面应用程序、Web 应用程序、移动设备等）。每个 Web 地图都由一个或多个 Web 地图服务组成，这些地图服务共同为用户提供有效的地图体验。

Web 地图通过交互式的地理信息展示，可以讲述故事以及回答问题。例如，可以找到或创建解决此问题的地图："美国有多少人居住在距离超市合理的步行距离或车程内?"。此 Web 地图包含显示了哪些住宅区在距离超市 10 分钟的车程或 1 英里的步行距离内的图层。为提供背景环境，此 Web 地图还应包含地形底图，其中包括城市、道路以及叠加在土地覆被上的建筑物和晕渲地貌影像。

制作 Web 地图与使用客户端 GIS 软件创建地图存在非常大的差异，主要表现在以下几个方面：

（1）在 Web 地图中，用户看到的所有信息都是通过网络由服务器发送到浏览器的，因此这就引入了延迟。

（2）在 Web 地图中，地图的内容可以同时来自几个不同的服务器。因此地图性能受到服务器的可访问性与速度的限制。

（3）在 Web 地图中，性能可能受到同时在线用户数量的影响。

（4）在 Web 地图中，用户体验同时也受到客户端应用程序显示技术的影响。该客户端应用程序通常就是网页浏览器。

这些考虑因素对于部分人来说显得很奇怪。例如，对于那些只使用过 QGIS 或 ArcMap 的人来说，肯定不习惯考虑带宽或与他人共享计算机。

更难的是，对于新的 Web 地图制作者们来说，最大的挑战在于了解在他们地图显示的数据的量，以及如何以亚秒级的速度获取在 Web 用户屏幕上绘制的所有信息。熟悉桌面 GIS 软件的人，习惯在地图中加入几十个甚至上百个图层，然后根据需要切换是否显示。强大的桌面计算机或许能够处理这类地图，但是，如果将该地图直接移动到互联网上，那么性能必然会是难以忍受的差。服务器需要宝贵的时间循环访问所有的图层，获取数据然后绘制，最后将图片返回到客户端。

针对这些问题，解决方案是根据不同的处理方法将图层归组。将那些只提供背景信息、地理框架信息的图层归为一组，并绘制为一个切片基础底图。相对于这个组，专题图层（地图中核心图层）作为一个或多个 Web 服务而叠加在底图上。此外，通常还需要包含一些与专

题图层交互的小组件，例如弹出式窗口、图表以及分析工具等。包含这 3 个部分的 Web 地图如图 2.3 所示。

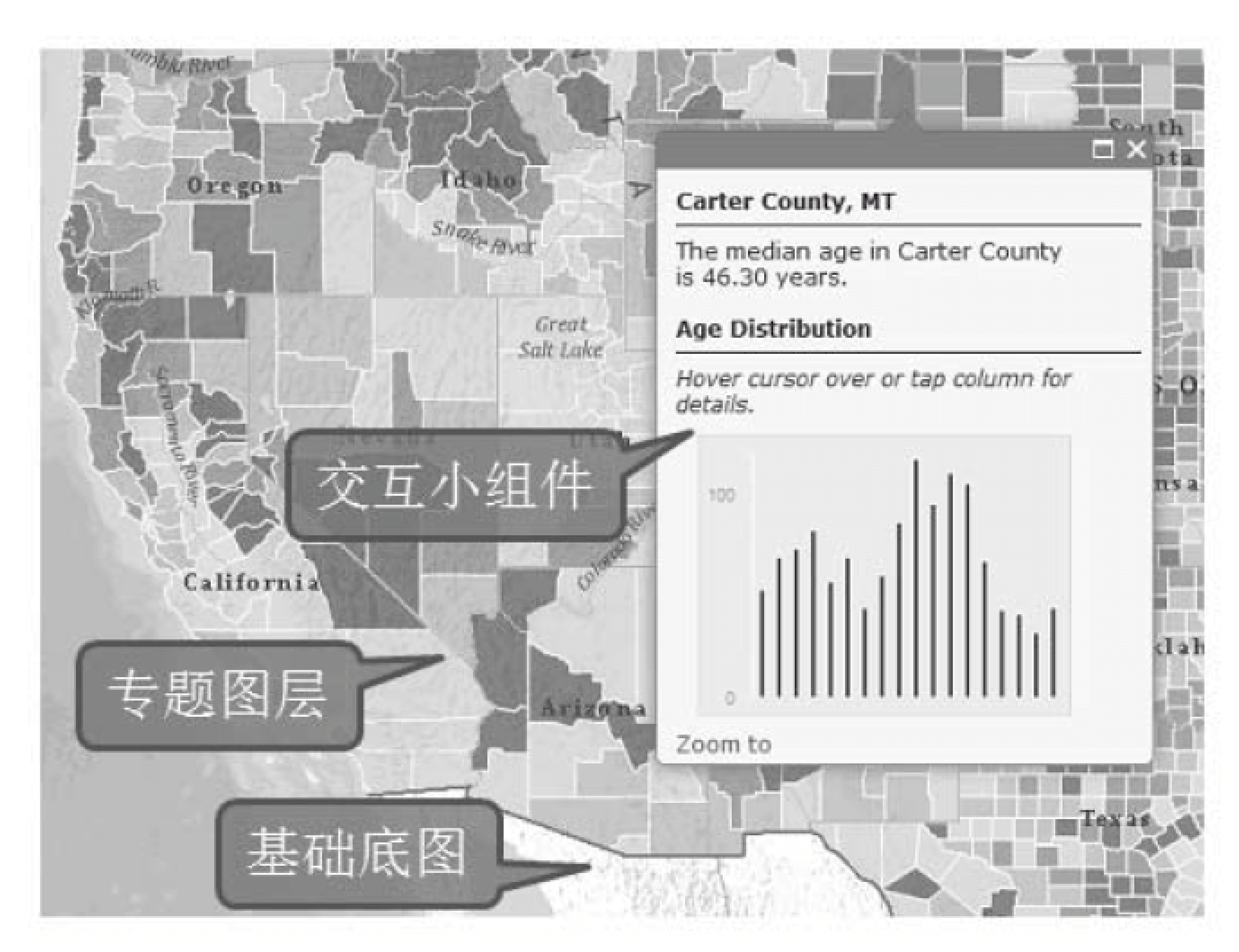


图 2.3　包含基础底图、专题图层与交互小组件的 Web 地图

下面将更详细地介绍这 3 个部分——基础底图、专题图层以及交互小组件。

2.2.1　基础底图

基础底图将重要数据组织起来，构成各种地图可重复使用的基础。底图为操作提供了基础，或者说画布。底图可用于常规用途，例如地形底图、影像底图或街道底图，还可用于某一特定主题，例如水文底图或地质底图。可在底图上绘制任何数据。底图提供了地理环境和参考详细信息。底图用于位置参考，并为用户提供叠加或聚合业务图层、执行任务以及可视化地理信息的框架。底图是执行所有后续操作和地图制图的基础，它为地理信息的使用提供了环境和框架。

虽然地图可能包含许多图层，例如道路、湖泊、建筑物等，通常会将这些图层栅格化为一系列的图像切片，并作为 Web 地图中一个图层来对待。这些切片地图包含了成千上万个预先绘制的图片，保存在服务器上，当用户漫游地图时，直接返回这些图片给用户。在第 5 章中将会更详细介绍切片地图。

有时使用两层切片地图来构成一个底图。例如，当使用航空影像作为切片地图图层时，由于没有注记，这时为了方便定位，在该切片地图上，再增加第二个由道路、地名等矢量数据组成的切片地图，共同组成底图。Google 地图就采用了这种方式。天地图将注记单独作为一切片图层。这样做的目的，一是可以使单独切片地图占用较少的磁盘空间，另一个是易于更新。

2.2.2 专题图层

专题图层也称为业务图层或操作图层，叠加在底图上，这也是用户访问地图的原因。由于大多数人并不关心专题图层，因此不能将其放置在底图中。但是一旦将其作为 Web 地图的专题图层，那么这些专题图层就是用户最感兴趣的。专题图层用于显示特定现象位置和分布的空间信息。如果地图标题为“北京市银行网点”，那么银行网点就是专题图层。如果地名标题“亚洲鸟类迁徙模式”，那么迁徙模式就是专题地图。

专题图层有时可以像底图一样，处理为地图切片，但是地图切片不适用那些变化迅速的数据。例如，如果需要在地图中显示警车的实时位置，这时就不能使用事先绘制好的切片地图了，而需要使用其他方式来绘制数据。这时可使用 WMS 等用于动态绘制地图的 Web 服务，来绘制专题图层。另一种方式是从服务器查询得到所有的数据，返回到客户端，然后在浏览器中绘制。这种方法还可以进一步使用弹出窗口等交互小组件来丰富页面。

专题图层与底图图层共同组成一个实际的 Web 地图。不过要特别注意的是，专题图层并不一定总是处于地图的最上层。专题图层可以处于两个切片地图之间。一个实例就是，最底层是地形要素底图，最上层是注记底图，中间是专题图层（图 2.4），可形象地称为“地图三明治”。

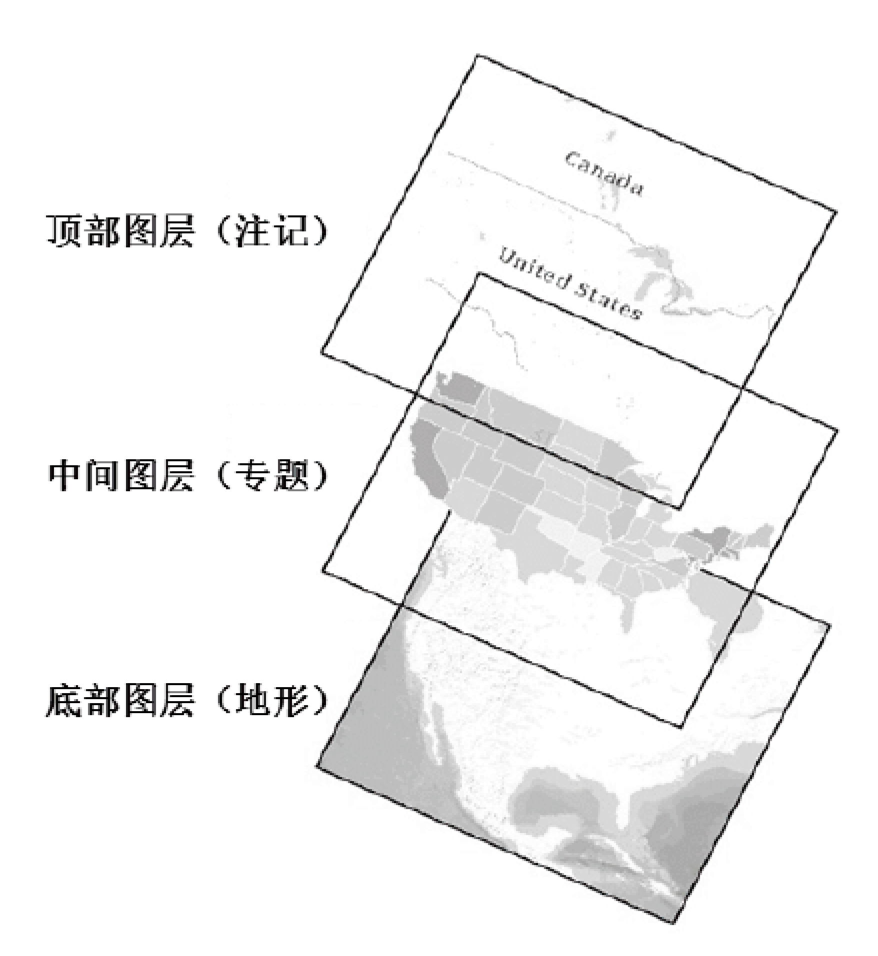


图 2.4　地图三明治

我国的天地图就是按照这种方式处理的，也就是将地形要素与注记分别处理为切片地图，共同组成为底图，以方便在其间插入专题图层，如图 2.5 所示。



图 2.5　天地图中同范围内的地形要素与注记切片地图

2.2.3　交互小组件

Web 地图中常常带有交互小组件，以便用户更进一步查询地图中的图层。例如，当用户单击某一地物要素时，会弹出一小窗口显示其信息，在页面的另一部分显示图表，地图中还显示一个时间轴以便切换数据显示年代等。一些 Web 地图还提供实时编辑 GIS 数据的功能，或向服务器提交地理处理任务，并在浏览器中绘制响应结果。一些 Web 地图提供空间分析功能，例如路径分析、可达范围、通视分析等。

这些交互小组件使得 Web 地图鲜活起来。要想使自己的 Web 地图有用，其中一个关键就是包含这些交互小组件，但是也只需要包含那些对用户最有用的，而不是提供一大堆的选项，或功能非常复杂，让用户望而却步。有时即使是一点点的进一步开发工作，例如在弹出窗口中使用用户友好的字段别名，就可能使得地图更有用、更可用。

交互小组件是需要编码工作的部分，其多少及其可使用性与 JavaScript 的编码量息息相关。不过，可以使用 OpenLayers 或 Leaflet 等这些开放的地图 API 来简化开发工作，此外，还可以使用更为基础的 JavaScript 类库，例如 Dojo、JQuery 等，来提供更基础的帮助。在后面的内容中会介绍 OpenLayers 及 Dojo 的使用。

2.3　实践 2：GeoServer 的安装与初步使用

GeoServer 是 OGC Web 服务器规范的 J2EE 实现，利用 GeoServer 可以方便地发布地图数据，允许用户对特征数据进行更新、删除、插入操作，通过 GeoServer 可以比较容易地在用户之间迅速共享空间地理信息。GeoServer 是自由及开源软件。

GeoServer 主要包含如下一些特点：

- 兼容 WMS 和 WFS 特性;
- 支持 PostGIS、Shapefile、ArcSDE、Oracle、VPF、MySQL、MapInfo;
- 支持上百种投影;
- 能够将网络地图输出为 JPEG、GIF、PNG、SVG、KML 等格式;
- 能够运行在任何基于 J2EE/Servlet 容器之上;
- 嵌入 MapBuilder 支持 AJAX 的地图客户端 OpenLayers。

（1）安装 Java。

由于 GoeServer 是基于 Java 开发的。因此在安装之前，必须确保安装了 Java。Java 下载地址为 https://www.java.com/zh_CN/。

（2）GeoServer 下载。

通过 www.geoserver.org 访问 GeoServer 的主页，单击“Download”按钮，进入下载页面。选择下载稳定版本（本书编写时稳定版本是 2.6.2），然后选择 Windows 安装程序。读者也可直接使用本书下载文件的 Tools 文件夹中的 geoserver-2.6.2.exe 文件。

（3）安装 GeoServer。

双击安装文件，所有设置都接受默认值。

（4）启动 GeoServer。

安装 GeoServer 之后，选择“开始>所有程序> GeoServer 2.6.2 > Start GeoServer”，启动的是一控制台应用程序，在其中显示一系列的启动状态信息，如图 2.6 所示。

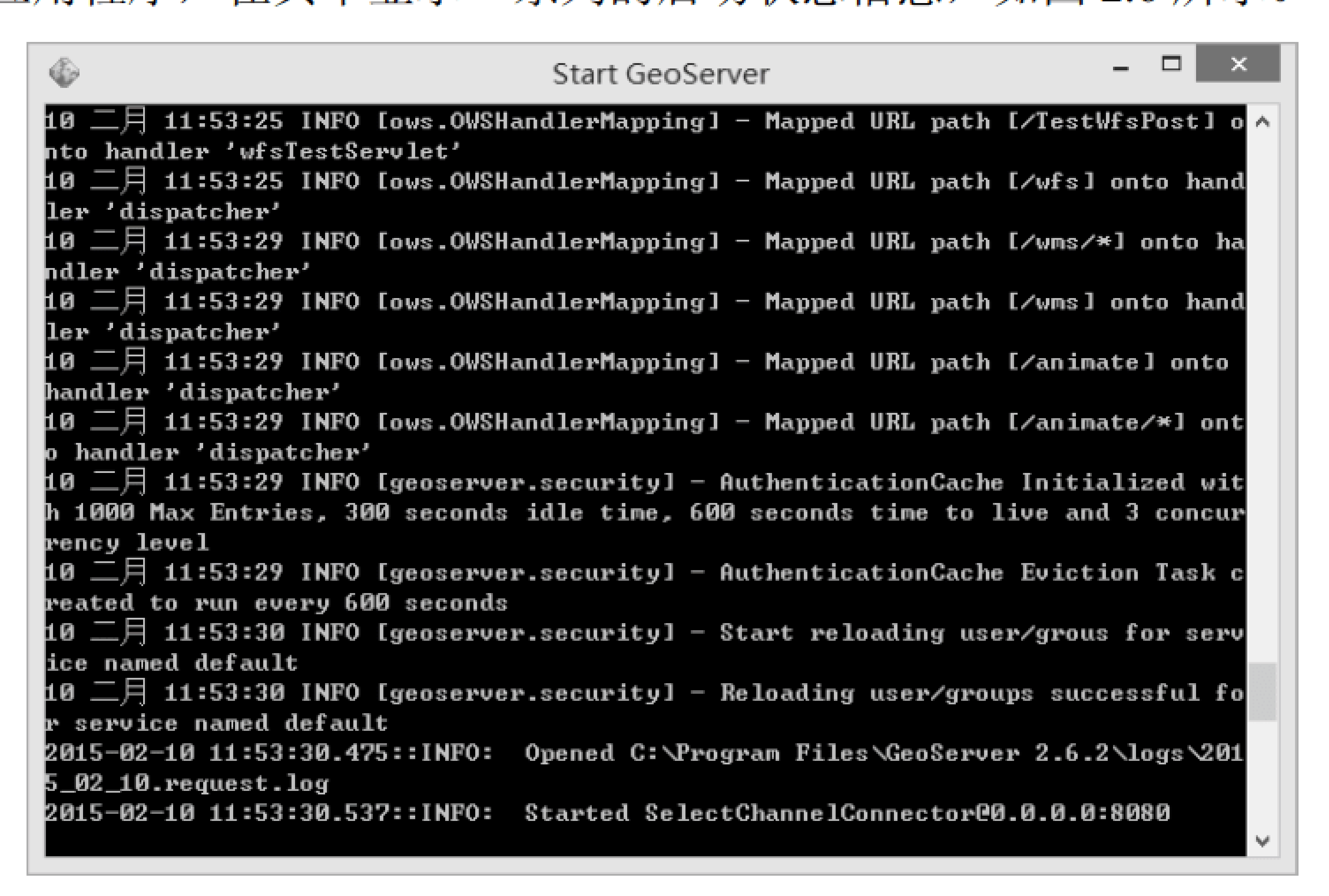


图 2.6　GeoServer 启动界面

稍等片刻，直到信息停止添加。必须保持本窗口打开，因为关闭该窗口就是关闭 GeoServer。当我们与 GeoServer 交互时，也会在该窗口中显示信息，这些信息可以帮助排除故障。

（5）打开 GeoServer 的 Web 管理页面。

GeoServer 的控制和管理是基于网页形式，所有和 GeoServer 相关的操作都要通过这个

Web 管理界面来进行，包括全局设置、数据发布与服务配置等。

通过选择“开始>所有程序> GeoServer 2.6.2 > GeoServer Web Admin Page”，或者直接在浏览器中输入“http://localhost:8080/geoserver/web/”地址，进入 GeoServer 的 Web 管理页面。通过该页面可以从安装了 GeoServer 的计算机或网络中其他计算机上管理 GeoServer。由于安装 GeoServer 时也同时安装了一个名为 Jetty 的 Web 服务器，默认设置其监听端口为 8080，因此该计算机能响应 Web 服务与页面的请求。

（6）登录 GeoServer 的 Web 管理页面。

在 GeoServer 的 Web 管理页面中输入用户名与密码进行登录。如果是默认安装，那么用户名为“admin”，密码为“geoserver”。

登录以后，可看到图 2.7 所示的页面。



图 2.7 GeoServer 的 Web 管理页面

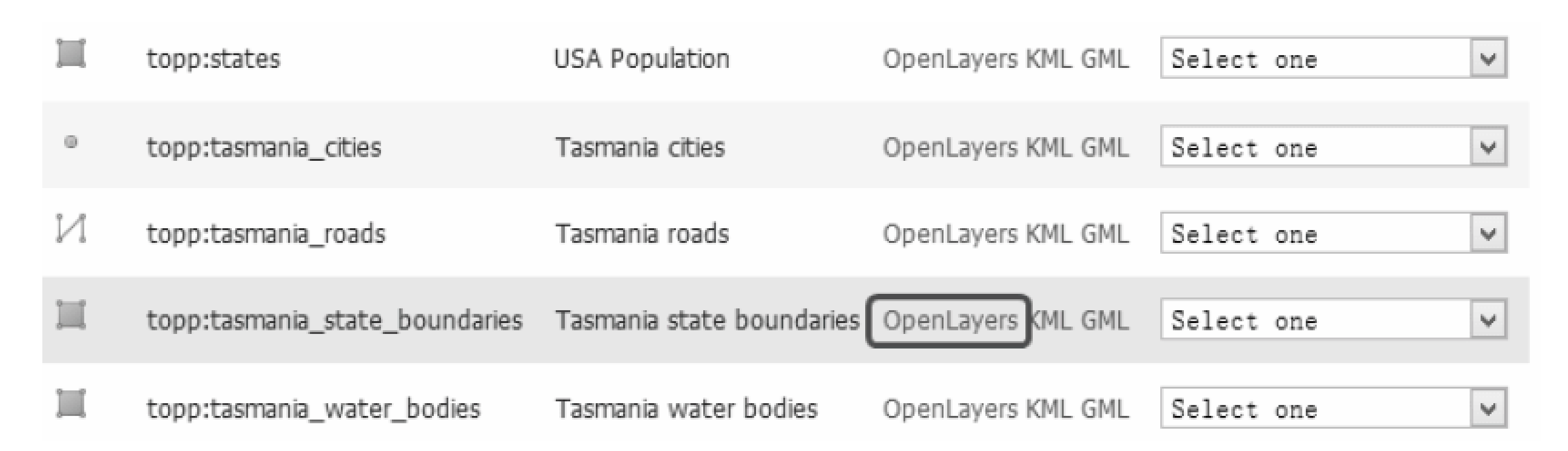
与 ArcMap、QGIS 将整个地图处理为.mxd 或.qgs 不同的是，GeoServer 使用的是图层与图层组的概念。将在服务器上准备发布为服务的数据定义为一组数据集，然后规定在发布为 Web 服务时的一些参数。

GeoServer 在安装后已经自带了一些样例图层与服务。我们先通过查看这些样例，来初步了解一下 GeoServer。

（7）图层预览。

在 GeoServer 的 Web 管理页面的左边菜单的“数据”部分，单击“Layer Preview”菜单

项，将在页面的右边部分列出了所有可预览的图层。向下滚动滚动条，滚动到 topp:tasmania_state_boundaries，如图 2.8 所示，然后单击 OpenLayers 链接。



	topp:states	USA Population	OpenLayers KML GML	Select one
	topp:tasmania_cities	Tasmania cities	OpenLayers KML GML	Select one
	topp:tasmania_roads	Tasmania roads	OpenLayers KML GML	Select one
	topp:tasmania_state_boundaries	Tasmania state boundaries	OpenLayers KML GML	Select one
	topp:tasmania_water_bodies	Tasmania water bodies	OpenLayers KML GML	Select one

图 2.8　在图层列表中单击“OpenLayers”连接进行图层预览

这会将地图显示为可漫游的 Web 服务。该服务满足 OGC 的 WMS 规范。地图的框架与漫游按钮都是基于 OpenLayers JavaScript 框架创建的。

此外，还可以从每行的最右边的下拉列表框中选择“WMS > OpenLayers”，实现同样的功能。仔细查看下拉列表框中的内容，了解 GeoServer 支持的不同输出格式。

关闭图层预览窗口，返回到 GeoServer 图层预览列表页面。这次单击“KML”连接，这会使用 KML 格式获取图层。如果安装了 Google Earth，那么将会使用 Google Earth 打开该图层。如果没有安装 Google Earth，那么可使用 Notepad 等文本工具打开该图层。

除了使用不同格式请求某图层，还可以请求一组图层。

在 GeoServer 图层预览列表页面，滚动到 Tasmania 图层组（绿色的正方形符号）行，单击“OpenLayers”连接，使用 OpenLayers 打开 Tasmania 图层组，如图 2.9 所示，可以看到 3 个图层都显示在地图中了。

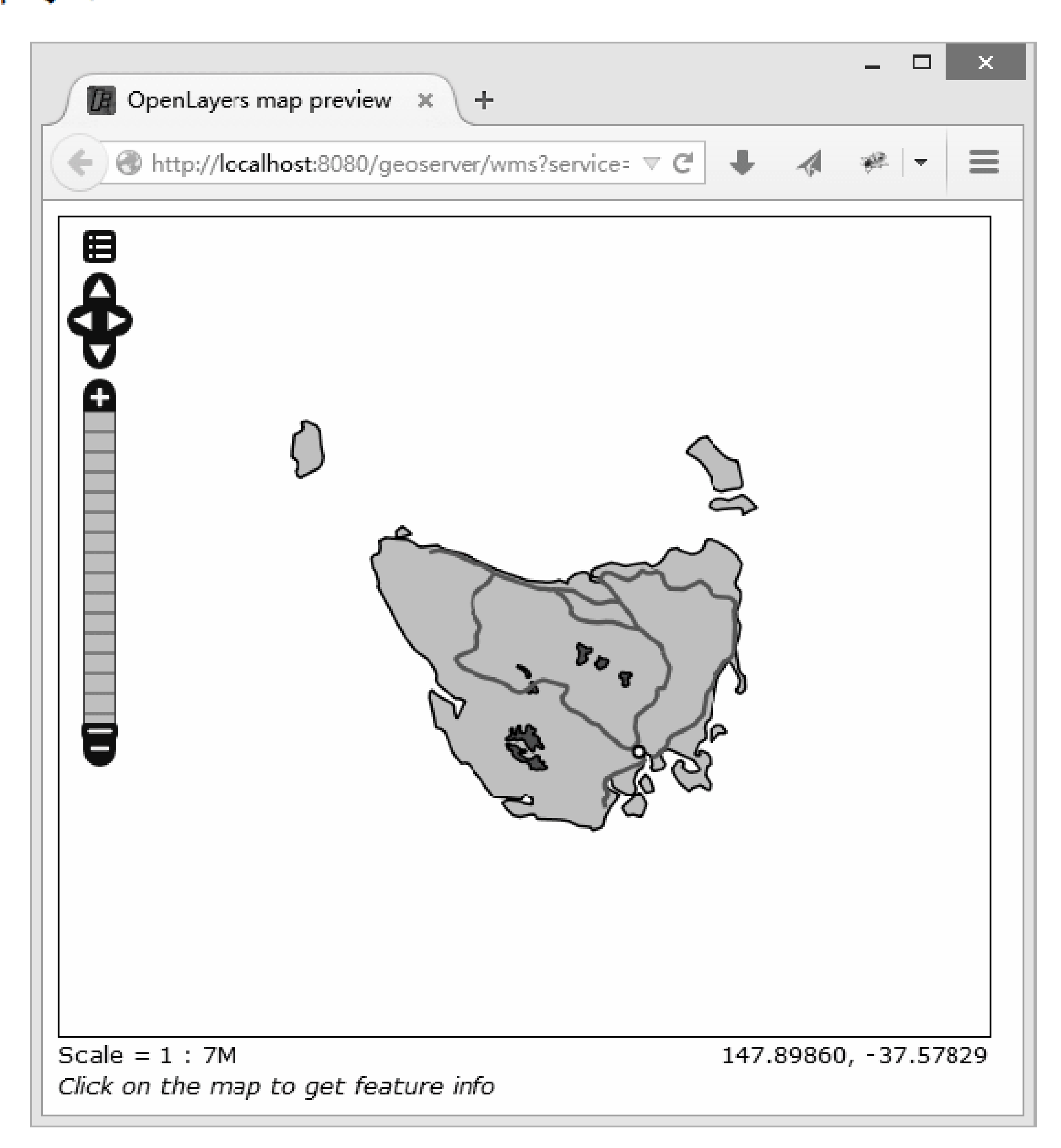


图 2.9　使用 OpenLayers 打开图层组

2.4　习　题

（1）利用 Firebug 等工具分析如下几个 Web GIS 网站：

- 北京公交网（www.bjbus.com/map）;
- 图吧（www.mapbar.com/search/）;
- 我要地图（www.51ditu.com）;
- “天地图•安徽”团购专题应用（ahcity.ahmap.gov.cn/tuangou/）。

回答如下问题：

- 在地图中区分基础底图、专题图层以及交互小组件;
- 查看服务器使用了哪些技术提供地图、专题图层以及交互小组件的，例如切片、动态生成图片还是浏览器端绘制矢量图形等;
- 查看使用了哪些软件及开发框架，分别是商业软件还是开源软件;
- 提出使网站运行更快与更实用的建议。

（2）熟悉 GeoServer 的 Web 管理页面内容与功能，并注意查看在 GeoServer 启动控制台窗口中显示的信息。

第 3 章

空间数据的存储与处理

从本章可以学习到：

- 空间数据常用的开放格式
- Web GIS 中的数据层
- 处理空间数据的开源工具
- 使用 QGIS 裁剪与投影变换矢量数据
- 使用 QGIS 处理栅格数据
- PostGIS 的安装与初步使用

在任何 Web GIS 应用的下方，都是显示在地图中表示地理实体的数据，当然这些数据除了展示在地图上的空间位置信息之外，还包含有大量的属性信息。那么如何存储与处理这些空间数据呢？本章将介绍在自由及开源软件领域存储与处理空间数据的多种选择。虽然受篇幅限制，并没有充分介绍数据库的理论及其设计，但是详细介绍了满足各种应用的多种数据格式。具体内容包括列出了空间数据常见的开放格式，各种数据存储结构和格式的优点。最后以实践的方式介绍了如何使用 QGIS 与 GDAL 来处理 GIS 数据，以及如何在 PostGIS 中创建空间数据库并导入空间数据。

3.1　空间数据常用的开放格式

Web 服务也是空间数据共享与互操作的一种重要方式，因此也能称为空间数据的一种格式，但是这里只介绍能独立存储在本地硬盘文件或空间数据的格式。

3.1.1　基于文件的数据

基于文件的数据包括 Shapefile、KML、GeoJSON 以及其他类型的文本文件。所有的矢量数据格式都有相应机制存储每个地理要素的几何位置与属性。此外，在 KML 等一些数据格式中还可以存储样式信息。这里介绍了最常用的基于文件的数据格式。

3.1.1.1　Shapefile

Shapefile 是 ESRI 开发的一种空间数据开放格式。目前，该文件格式已经成为了地理信息软件界的一个开放标准，也是一种重要的矢量数据交换格式。

Shapefile 文件格式实际上是由多个文件组成的。其中，要组成一个 Shapefile，有 3 个文件是必不可少的，它们分别是.shp、.shx 与.dbf 文件。表示同一数据的一组文件其文件名前缀应该相同。例如，存储一个关于湖的几何与属性数据，就必须有 lake.shp、lake.shx 与 lake.dbf 这 3 个文件。而其中“真正”的 Shapefile 的后缀为 .shp，然而仅有这个文件数据是不完整的，必须要把其他两个附带上才能构成一组完整的地理数据。除了这 3 个必需的文件以外，还有 8 个可选的文件，使用它们可以增强空间数据的表达能力。此外，所有的文件都必须位于同一个文件夹之中。

3.1.1.2　KML

KML 是一种基于 XML 的格式，用于存储地理数据和相关内容。KML 文件要么以.kml 为扩展名，要么以.kmz（表示压缩的 KML 文件）为扩展名。

KML 由 Keyhole 公司开发，此后该公司被 Google 收购。2008 年，Google 自愿将 KML 提交给 OGC，宣布不再控制 KML 标准，而移交给 OGC 去维护发展，从而成为 OGC 的一种

官方标准。

KML 可以由要素和栅格元素组成，这些元素包括点、线、面和影像，以及图形、图片、属性和 HTML 等相关内容。尽管通常将数据集视为独立的同类元素（例如，点要素类只能包含点，栅格只能包含像元或像素，而不能包含要素），但单个 KML 文件却可以包含不同类型的要素，并可包含影像。

由于 KML 是单个的高度可移植文件，可包含图层的全部内容和要素几何、影像、符号系统、描述、属性等地图元素，以及其他相关内容，并且可通过许多受欢迎的免费应用程序进行查看，如 Google 地球和 ArcGIS Explorer，所以 KML 成为与广大公众共享地理数据的极佳格式。现在很多 GIS 相关企业也追随 Google 开始采用此种格式进行地理数据的交换。

3.1.1.3 GeoJSON 与 TopoJSON

JSON 是一种轻量级的数据交换格式，易于用户阅读和编写，同时也易于机器解析和生成。它基于 JavaScript 编程语言。 JSON 采用完全独立于语言的文本格式，但是也使用了类似于 C 语言家族的习惯（包括 C、C++、C#、Java、JavaScript、Perl 和 Python 等）。 这些特性使 JSON 成为理想的数据交换语言。JSON 比 XML 更小、更快、更易解析。

JSON 建构于两种结构：

（1）“名称/值”对的集合。在不同的语言中，它被理解为对象、记录、结构、字典、哈希表、有键列表或者关联数组。

（2）值的有序列表。在大部分语言中，它被理解为数组。

JSON 举例如下：

```
{
    "firstName": "John",
    "lastName": "Smith",
    "sex": "third",
    "age": 25,
    "address":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber":
    [
        {
          "type": "home",
          "number": "212 555-1234"
        },
```

```
            {
              "type": "fax",
              "number": "646 555-4567"
            }
        ]
    }
```

按照这种模式，GeoJSON 利用 JSON 来存储矢量地理要素，只是对各名称做了规范。GeoJSON 支持点、线、面、多点、多线、多面和几何集合等几何类型。GeoJSON 里的要素包含一个几何对象和其他属性，要素集合表示一系列要素。

可以将 GeoJSON 保存为.js 文件，这样便可以在 Web GIS 中直接引用。此外，当前主流的 Web 服务通常的响应也是 GeoJSON 格式。

TopoJSON 是 GeoJSON 简化后的版本。TopoJSON 与 GeoJSON 相比，文件大小缩小了 80%，这是因为：

- 边界线只记录一次（例如对于中国广西省和广东省的交界线只记录一次）;
- 不使用浮点数，只使用整数。

因此将 GeoJSON 转换为 TopoJSON 数据，对于解决因文件过大而造成的读取速度过慢是相当简单、有效的。不过当前 TopoJSON 还只在 D3.js 中有比较广泛的使用，还不是世界范围内认可的格式。

3.1.1.4　其他文本文件

其他矢量数据文件的格式还有.gpx、包含空间坐标信息的.csv 等。

GPX（GPS eXchange Format，GPS 交换格式）是一个 XML 格式，为应用软件设计的通用 GPS 数据格式。它可以用来描述路点、轨迹、路程。这个格式是免费的，可以在不需要付任何许可费用的前提下使用。它的标签保存位置、海拔和时间，可以用来在不同的 GPS 设备和软件之间交换数据。如查看轨迹、在照片的 exif 数据中嵌入地理数据。

在 GPX 中，一个没有顺序关系的点集合叫作路点。一个有顺序的点的集合叫作轨迹或者路程。轨迹是一个人曾经走过的记录，路程是建议的下一步要走的地方。所以一般来讲，轨迹里的点包含时间信息，路程里的点没有时间信息。

最小的一个 GPX 文件，仅仅包含一个经纬度坐标的点，其他的都是可选的。

CSV（Comma-Separated Values，逗号分隔值）文件以纯文本形式存储表格数据（数字和文本），纯文本意味着该文件是一个字符序列。CSV 文件由任意数目的记录组成，记录间以某种换行符分隔；每条记录由字段组成，字段间的分隔符是其他字符或字符串，最常见的是逗号或制表符。通常，所有记录都有完全相同的字段序列。第一条记录可以是字段名。

3.1.1.5 栅格格式

大多数栅格数据格式都是开放的，包括 JPEG、PNG、TIFF、BMP 等。过去 GIF 曾使用了专利压缩格式，但在 2003 年 6 月过期了。

此外，切片地图以及 WMS 等 Web 服务返回的也都是栅格格式的图片。KML/KMZ 文件也能包含栅格数据。

3.1.2 基于空间数据库的数据

当数据集越来越多并越来越复杂时，最好将数据存储在数据库中。这通常使得更容易运行高级查询、建立关系数据集，并管理编辑的数据。数据库也可以提高性能，并引入执行空间操作工具。

商业的数据库包括微软的 SQL Server、Oracle Spatial 以及 ArcSDE 中间件。通过 ArcSDE 中间件，可以连接包括 FOSS 数据库在内的多种数据库。最为常用的 FOSS 数据库有 PostGIS 与 SpatiaLite 等，下面分别介绍这两个数据库。

3.1.2.1 PostGIS

PostGIS 是一个开源程序，它为关系型数据库 PostgreSQL 提供了存储空间地理数据的支持，使 PostgreSQL 成为了一个空间数据库，能够进行空间数据管理、数量测量与几何拓扑分析。PostGIS 实现了 OGC 所提供的简单要素的 SQL 实现参考。

PostGIS 实现了一个基于轻量级的几何体实现，并提供了完善的索引，这大大减少了硬盘与内存的存储量。轻量级的几何体实现使服务器能够把磁盘中更加大量的数据载入到内存之中，这大大提高了查询的性能。

PostGIS 已经注册成为了 OGC 的简单要素的 SQL 标准的其中一种实现。然而由于某些原因，OGC 并未把 PostGIS 列为一种“兼容”的实现。其原因包括 PostGIS 扩展了 WKB 与 WKT 格式来存储带有三维或四维坐标的几何体，该扩展并不符合 OGC 的最新定义。

在 Windows 平台下，PostGIS 提供了一个 pgAdminII 的插件，该插件能够把 ESRI Shapefile 格式的地理数据导入 PostGIS 数据库之中。

大多数 FOSS GIS 软件都提供了连接 PostGIS 数据库的接口，例如 QGIS、uDig 以及 GeoServer 等。常用商业 GIS 软件也都有接口，例如 ArcGIS、MapInfo 等。

3.1.2.2 SpatiaLite

SpatiaLite 是一套具有空间数据功能的 SQLite 数据库系统。

正如其名称所显示的，SQLite 是一种轻量级的数据库引擎。但是 SQLite 并没有使用“客户端/服务器”结构，也并不需要安装关系数据库管理系统，因此 SQLite 数据库可以非常容易复制，并运行在各种类型的设备上。因此，SpatiaLite 非常类似 ESRI 产品中的文件地理数据

库（File Geodatabase）。而且相对 MySQL、PostgreSQL 这两款开源的世界著名数据库管理系统来讲，它的处理速度比它们都快。

SpatiaLite 实现了 OGC 所提供的简单要素的 SQL 实现参考。虽然 SQLite 本身也具备 R 树索引以及几何类型，但是要进行高级的空间查询以及支持多种地图投影，则必须使用 SpatiaLite。SpatiaLite 提供了一软件库以及几个实用工具。这些工具包括命令行工具（在 SQLite 中加入了空间宏）、一个操作数据的图形化界面以及一个用于浏览的数据简单桌面 GIS 工具。

由于只是一个二进制的文件，因此 SpatiaLite 常用来作为交换空间数据的矢量格式。

SpatiaLite 虽然现在还不如 PostGIS 成熟，但是也越来越得到更广泛使用。QGIS 也能连接 SpatiaLite。

3.2　Web GIS 中的数据层

在第 2 章中，介绍了在 Web GIS 应用中包含一个数据层。该层可以很简单，就是存储在计算机中某个文件夹下的几个 Shapefile 文件，也可以是包含几台企业级服务器的很复杂的生态系统，既包含独立文件，也包含关系数据库。

数据层存储了 Web GIS 使用的数据集。几乎可以肯定的是，其中包含专题图。如果决定创建自己的底图与切片集，那么该层还包含了底图图层数据。不过，也可使用其他 GIS 服务器的底图或一些专题图层，以免维护与更新数据。

一些单位或组织出于安全与性能的考虑，不会将每天都在编辑与更新的数据放到网络上。如果允许网络地图用户修改数据库，必须考虑避免数据被删除、损坏或破坏。同时，也不希望由于外部用户的大量使用而导致自己内部 GIS 用户的响应变慢，反之亦然。

出于这些考虑，因此通常复制一份数据库专门用于互联网应用。如果这些要素数据不需要让终端用户编辑，那么该复制的数据库就是只读的。如果需要让用户编辑，那么则需要使用自动脚本或 Web 服务让两个数据库同步。

3.2.1　服务器的选择

通常可以通过最大限度地减少存储数据的服务器与最终用户之间的中转站数量，来增加网站地图的性能。如果数据是文件形式或简单的数据库，那么可以将数据直接存储在 GIS 服务器上，而不需要再使用单独的数据库服务器，这样便减少了 GIS 服务器与数据库服务器之间的网络传输。但是，当数据量非常大的时候，或使用数据库的用户数量很多时，最好将数据库单独安装在独立的服务器上。这样便可以更加专注备份流程和冗余存储机制，防止数据丢失和损坏。也有助于许多并发用户访问时，防止数据库和服务器资源的竞争。

当将数据单独存储在数据库服务器上时，确保防火墙不会阻止机器间在所有必要端口之间的通信。这可能需要涉及咨询 IT 部门的员工。同时还需要确保运行 Web 服务的进程被允许从其他计算机中读取数据。此外，不能使用像“C:\Data\China”这种本地路径来访问数据

集，而是需要使用共享文件夹的网络名称，例如“\\DataServer\data\Japan”。

3.2.2 文件与数据库方式的选择

当设计数据层时，需要确定是将数据存储为一系列的简单文件，例如 Shapefile 或 KML 等；还是存储到支持空间数据的数据库中，例如 PostGIS 或 SpatiaLite。文件存储方式适用于那些不经常变换的，而且数据量相对比较小的数据集。文件方式存储数据当然要比数据库存储简单许多，而且便于在用户与计算机之间迁移和共享。

如果需要存储大量的数据，或者数据需要被不同的部门经常编辑，或需要维护关系表来链接数据集，那么使用数据库方式更合适。数据库可以提供强大的 SQL 查询以及计算空间关系（例如相交、包含等）。

如果已经有安装在数据库中的长期运行的 GIS 项目，现在需要将其发布到网络，那么就需要针对上述利弊来权衡是否保留数据库或提取数据并将其复制到基于文件的数据集中。

3.2.3 开放数据格式与专有格式的选择

在前面的内容中已经介绍了开放的数据格式，例如 Shapefile、KML、JPEG 等，并介绍了各自优劣。开放数据格式所对应的是专有数据格式，是由特定的软件供应商创建的，没有公开的文档说明，或不能由任何其他开发人员创建或扩展。ESRI 的文件地理数据库就是一种典型的专有格式。虽然最近 ESRI 提供了用于创建文件地理数据库的 API，但是不能扩展或还原其底层格式。

一些更广泛应用的开放数据格式实际上也是由商业软件厂商设计的，因为某些原因，他们决定将格式开放。两个典型例子是 ESRI 的 Shapefile 与 Adobe 的 PDF。虽然将数据格式开放可能会引入 FOSS 同类软件的竞争，但是同时也增强了其商业软件的互操作性。并且，如果数据格式得到广泛地采用与推广，必然会增强供应商在软件社区的影响力和公信力。

3.3 处理空间数据的开源工具

空间数据的采集、处理是建立 Web GIS 的第一步。在大多数情况下，收集到的基础空间数据都不可能完全满足需要，这时需要对这些数据进行必要的处理。例如，根据 DEM 创建坡度、坡向、坡度分带等；空间插值、投影变换等。实际上，在将数据集成到 Web GIS 中之前，对于大部分的数据都需要进行一定类型的处理。

下面简单介绍几个本书实践过程中会用到的数据处理开源软件与工具。

3.3.1　QGIS

QGIS 提供了很多常用的矢量和栅格处理工具。此外，还有很多开发者在 QGIS 用户社区贡献的插件，可以扩展 QGIS 功能。

打开 OGIS 软件，在“矢量”菜单中包含了许多菜单项用于处理矢量数据，包括合并、缓冲和裁剪等，如图 3.1 所示。



图 3.1　QGIS 中的矢量数据处理功能

此外，另一些强大的功能隐藏在了矢量图层的“另存为”右键菜单中。通过“矢量图层另存为”对话框，可以在不同的格式间进行转换，例如将 Shapefile 转换为 GeoJSON，还可以进行投影转换等，如图 3.2 所示。



图 3.2　通过“另存为”实现格式与投影的转换

在 QGIS 中的“栅格”菜单内包含了许多处理栅格数据的功能，包括插值、等高线、地形分析、分区统计等。

如果在 QGIS 中没有找到某处理功能，那么这时可使用其他开发人员开发的插件。QGIS 在安装时也已经安装了一些有用的插件。选择“插件”菜单中的“管理并安装插件”菜单项，切换到“已安装”面板便可查看已安装的插件。在该面板中还可以卸载某些不需要的插件。

如果想增加某些插件，可先在该对话框的“全部”面板中选择某插件，查看其介绍，如图 3.3 所示，如果正是所需要的，单击“安装插件”按钮即可。



图 3.3　查看并安装插件

3.3.2　GDAL 与 OGR 工具

虽然大多空间数据处理函数使用了最主流的算法，而且这些算法大多都有详细的描述文档，但是如果每个开源软件的开发人员对这些相同的操作都从头编码，那么必然会导致大量额外的烦琐工作，并且可能会引入错误与不一致。因此许多 GIS 开源软件就充分使用 GDAL（Geospatial Data Abstraction Library）这一开源的代码库，来执行最常见的功能。

GDAL 是一个在 X/MIT 许可协议下的开源栅格空间数据转换库。它利用抽象数据模型来表达所支持的各种文件格式。它还有一系列命令行工具来进行数据转换和处理。OGR 是 GDAL 项目的一个分支，功能与 GDAL 类似，只不过它提供对矢量数据的支持。有很多著名的 GIS 类产品都使用了 GDAL/OGR 库，包括 ESRI 的 ArgGIS、Google Earth、GRASS GIS 和 QGIS 等系统。

使用 GDAL 与 OGR 大致有 3 种途径，一种是直接使用 QGIS 等图形界面的软件；第二种是使用 Python、C#、Java 等开发语言调用库中的函数；复杂性与灵活性居于上述两者之间的是第三种，就是使用命令行工具来调用。当安装 QGIS 时，同时安装了这些工具。在本章的实践部分会介绍其中一些工具的使用方法。

此外，还有许多其他 FOSS 工具用于处理空间数据，而且还会不时涌现出新的工具。

3.4　实践 3：使用 QGIS 裁剪与投影变换矢量数据

本实践将首先介绍如何使用 QGIS 来裁剪并投影变换矢量数据，然后介绍如何使用 OGR 命令行的功能实现同样的目的。使用命令行的好处就是可以对整个文件夹下所有的文件进行循环处理。

本实践使用的数据位于下载文件的“Data\PhiladelphiaBaseLayers”文件夹中，数据属于美国宾夕法尼亚州东南部港口城市费城。数据主要是从 OpenStreetMap 中下载并提取的。

3.4.1　使用 QGIS 裁剪数据并转换投影

（1）将数据复制到一个简单的文件夹下，我们假设为“C:\Data\PhiladelphiaBaseLayers”。

（2）了解原始数据。

在 PhiladelphiaBaseLayers 文件夹中存放了一大堆 Shapefile 文件，还有 3 个用于练习的文件夹，分别是 clipFeature、clipped 与 clippedAndProjected。

数据集使用的是地理坐标系统，覆盖范围是费城。本实践的任务是将数据裁剪到费城城市边界范围之内，然后将它们投影到当前主流的在线的 Web 地图使用的 Web 墨卡托投影坐标系统中。

Google 地图于 2005 年首先使用了 Web 墨卡托投影，但是最初没有得到欧洲石油测绘组（European Petroleum Survey Group，简写为 EPSG）的认可，所以没有正式的空间引用标识符（Spatial Reference Identifier，SRID），只有一个非正式代码，即 900913。在 2008 年，EPSG 提供了一正式的标识（EPSG:3785）与正式的名称（Popular Visualization CRS / Mercator），同年晚些时候，EPSG 将其标识更新为 EPSG:3857，名称更新为“WGS 84 / Pseudo-Mercator”。其他标识符还有 ESRI:102113、ESRI:102100、OpenLayers:900913 与 OSGEO:41001 等，其中 ESRI:102113 对应 EPSG:3785，而 ESRI:102100 对应 EPSG:3857。当前几乎所有的主要在线地图，包括 Google 地图、必应地图、MapQuest、MapBox 与 OpenStreetMap 等，都采用该投影坐标系统。

（3）裁剪矢量数据。

打开 QGIS，加入 fuel.shp 与 clipFeature/city_limits.shp 两个图层。然后选择“矢量>地理处理工具>Clip”菜单项，打开“裁剪”对话框。将“输入矢量图层”设置为 fuel，“裁剪图层”为 city_limits，并将“输出 shape 文件”设置为 clipped 子文件夹的 fuel.shp 文件，如图 3.4 所示。



图 3.4　裁剪矢量图层设置对话框

然后单击“确定”按钮，执行裁剪操作。裁剪完成后，QGIS 自动将裁剪后的图层加入到地图中，可以看到新图层只包含着费城边界范围之内的要素。

（4）投影变换。

在图层控制器中，单击右键以选择裁剪后的 fuel 图层，然后选择“另存为”菜单项，打开“矢量图层另存为”对话框，确保输出格式设置为 ESRI Shapefile 文件，将“另存为”设置为 clippedAndProjected 子文件夹下的 fuel.shp。

单击“坐标参照系”旁的“更改”按钮，打开“坐标参照系选择器”对话框，在“过滤”文本框中输入“pseudo”，在“世界坐标参照系”中列出了过滤后的结果，选择“WGS 84 / Pseudo Mercator”，如图 3.5 所示。



图 3.5　选择坐标参照系

选择了坐标系统后，单击“确定”按钮返回到“矢量图层另存为”对话框中，再次单击“确定”按钮。这次 QGIS 没有将另存后的数据加入到当前地图中，要核实是否按照要求进行了坐标转换，最好是从新地图中加入该数据。

在 QGIS 中新建一个项目，在地图中加入 clippedAndProjected/fuel.shp。显示效果如图 3.6 所示。

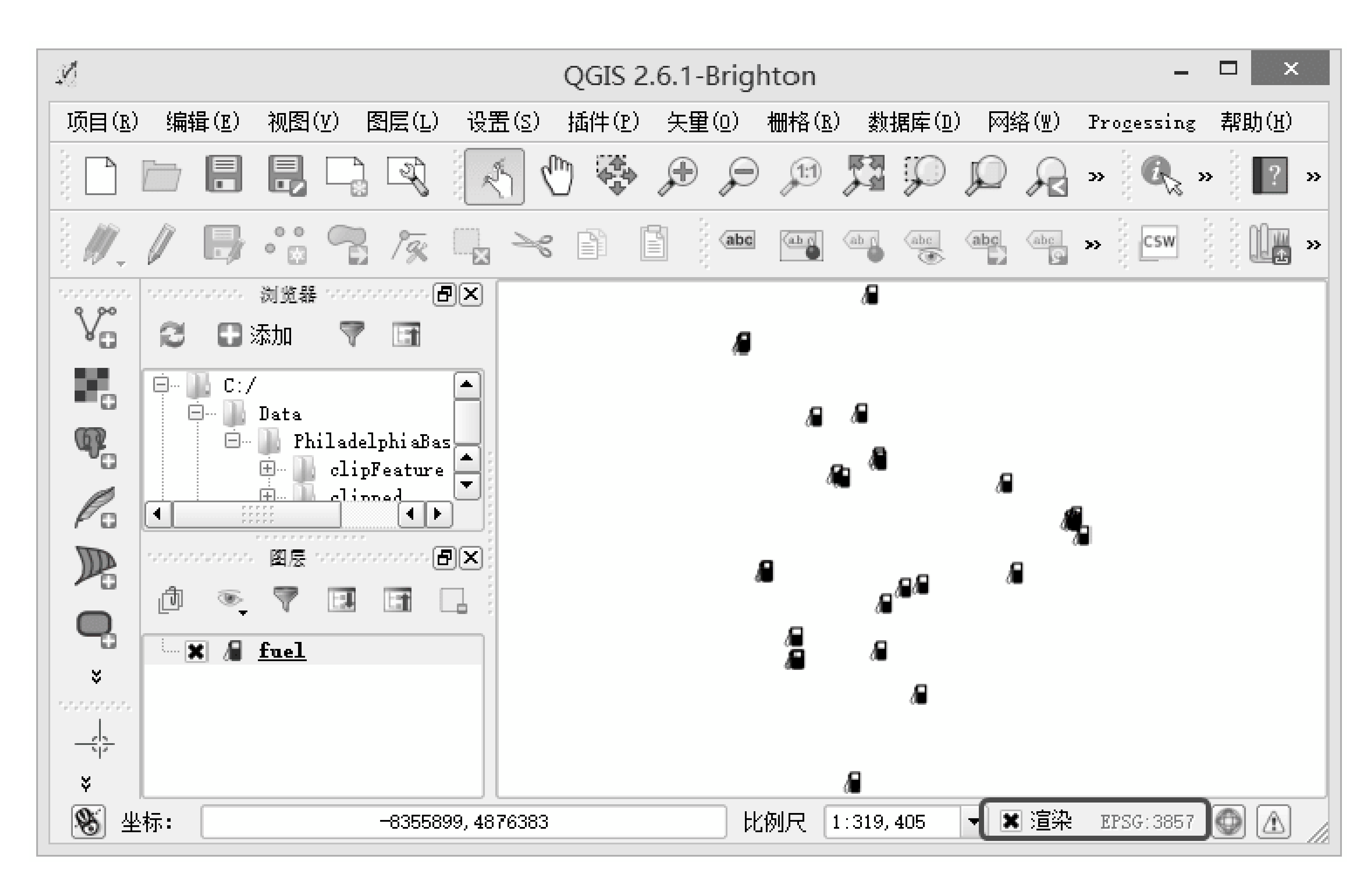


图 3.6　确认投影转换成功

与 ArcMap 一样，在 QGIS 中，第一个加入地图的图层决定整个地图的投影。注意在 QGIS 窗口的右下角显示了 EPSG 的代码为 3857，证明数据确实经过了投影。以后加入的图层也都会使用该投影来显示。

3.4.2　使用 OGR 命令行工具裁剪与投影变换数据

上述操作很简单。但是如果一次处理几十个或上百个图层，那么这种方法既麻烦，又耗时，且容易引起错误。对于这类情况，可以使用 OGR 的命令行工具来实现。

在安装 QGIS 时，也一同安装了运行 GDAL 与 OGR 功能的命令行工具，而且在桌面还有一个快捷方式，即 OSGeo4W。

（1）运行 OSGeo4W。

在桌面上双击 OSGeo4W 快捷方式，如果没有该快捷方式，可选择“开始>所有程序>QGIS>OSGeo4W”，打开一个空的命令行窗口。

（2）裁剪数据。

我们需要使用 ogr2ogr 工具来实现裁剪与投影变换。虽然 OGR 可以用一个命令同时完成裁剪与投影变换工作，但是在其执行过程中先执行投影变换。对于本实践，由于原始数据覆盖范围大得多，所以先裁剪再投影要更合适。

在“www.gdal.org/ogr2ogr.html”网页详细介绍了 ogr2ogr 的使用。在使用该工具时，首先提供一些可选参数，然后再提供一些必选参数。第一个必选参数是输出数据集的名称，第

二个必选参数是输入数据集的名称。

在控制台窗口中输入并执行如下命令：

ogr2ogr-skipfailures-clipsrc c:\data\PhiladelphiaBaseLayers\clipFeature\city_limits.shp c:\data\PhiladelphiaBaseLayers\clipped\roads.shp c:\data\PhiladelphiaBaseLayers\roads.shp

由于道路数据非常多，因此需要稍等片刻。完成之后，在 QGIS 中重新创建一项目，将该 clipped\roads.shp 数据加入到地图中，确保所有的道路要素在费城城市范围线之内。

仔细查看上述命令中的参数，其中包含了两个可选参数，一个是-skipfailures，这对于 OpenStreetMap 数据很有用，因为可能存在一些奇怪的拓扑。第二个可选参数是-clipsrc，代表裁剪要素。最后两个参数分别代表输入与输出数据集。

（3）投影变换。

在控制台窗口中输入并执行如下命令：

```
ogr2ogr -t_srs EPSG:3857 -s_srs EPSG:4326 c:\data\PhiladelphiaBaseLayers\clippedAndProjected\roads.shp c:\data\PhiladelphiaBaseLayers\clipped\roads.shp
```

等命令执行完成之后，在 QGIS 中重新创建一项目，将该 clippedAndProjected \roads.shp 数据加入到地图中。可发现比投影前的数据在垂直方向拉伸了一些。

在上述命令行中，-s_srs 参数指定源数据集的坐标系统（EPSG:4326，即 WGS 1984 地理坐标系统），-t_srs 参数指定目标坐标系统（EPSG:3857，即 Web 墨卡托投影）。最后两个参数同样分别代表输入与输出数据集。

如果不清楚源数据所使用的投影及其对应的标识符，可以在 QGIS 中新建一个项目，将数据加入到地图中，在 QGIS 窗口的右下角便可显示其标识符，如图 3.6 所示。如果不清楚目标投影的标识符，那么可以在 QGIS 中使用“另存为”菜单项来查看。

（4）使用循环命令。

通过上面的操作，可以看到 ogr2ogr 非常方便。其实更实用的是其自动功能。

删除 PhiladelphiaBaseLayers 文件夹中 clipped 与 clippedAndProjected 两个子文件夹中的所有文件。

在 OSGeo4W 控制台窗口中输入如下命令，切换操作路径：

```
cd c:\data\PhiladelphiaBaseLayers
```

输入如下命令，对所有该文件夹中所有的数据进行裁剪：

```
for %X in (*.shp) do ogr2ogr -skipfailures -clipsrc c:\data\PhiladelphiaBaseLayers\clipFeature\city_limits.shp c:\data\PhiladelphiaBaseLayers\clipped\%X c:\data\PhiladelphiaBaseLayers\%X
```

通过控制台窗口输出的消息，可以看出 ogr2ogr 对文件夹中的所有数据集进行循环处理。

该命令与之前的命令很相似，只是使用了用“%X”表示的一个变量，代替具体的数据集名称。而且还同时使用了循环查找文件夹中所有的 Shapefile 并执行裁剪命令。

在控制台窗口中输入如下命令，将操作路径切换到 clipped 文件夹中：

cd c:\data\PhiladelphiaBaseLayers\clipped

在控制台窗口中输入如下命令，对所有数据集进行投影变换，并保存在 clippedAndProjected 文件夹中：

```
for %X in (*.shp) do ogr2ogr -t srs EPSG:3857 -s srs EPSG:4326 c:\data\PhiladelphiaBaseLayers\clippedAndProjected\%X c:\data\PhiladelphiaBaseLayers\clipped\%X
```

通过 QGIS 确保命令确实得到正确执行，如图 3.7 所示。

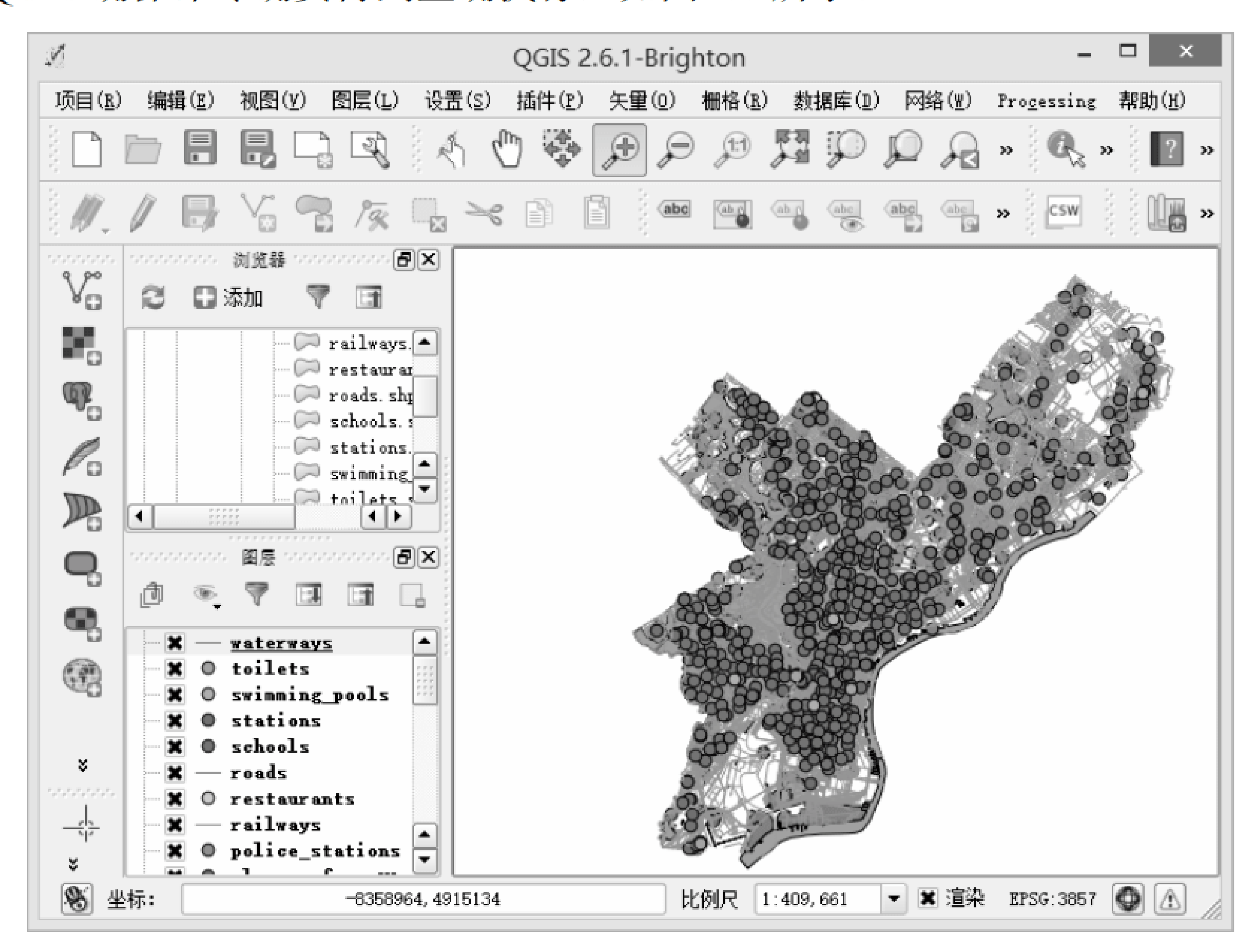


图 3.7　经过裁剪与投影变换的所有数据

3.4.3　在批处理中运行 OGR 功能

如果在将来还需要进行同样的操作，那么可以将命令放置在批处理文件中。批处理文件就是扩展名为.bat 的文本文件。

使用任意一种文本编辑器，在其中输入如下内容：

```
cd /d C:\Data\PhiladelphiaBaseLayers
set ogr2ogrPath="c:\program files\QGIS Brighton\bin\ogr2ogr.exe"
for %%X in (*.shp) do %ogr2ogrPath% -skipfailures -clipsrc D:\DataExamples\PhiladelphiaBaseLayers\clipFeature\city limits.shp D:\DataExamples\PhiladelphiaBaseLayers\clipped\%%X D:\DataExamples\PhiladelphiaBaseLayers\%%X
for %%X in (*.shp) do %ogr2ogrPath% -skipfailures -s_srs EPSG:4326 -t_srs EPSG:3857 D:\DataExamples\PhiladelphiaBaseLayers\clippedAndProjected\%%X D:\DataExamples\PhiladelphiaBaseLayers\clipped\%%X
```

如果读者没有完全按照本书设置数据目录或 QGIS 的安装目录，那么需要按照读者的实际情况修改上述命令中的路径。

将文件保存为 clipAndProject.bat。然后在 Windows 资源管理器中双击该文件，便可执行裁剪与投影变换工作。

3.4.4 数据整合

在后面的内容中需要使用到裁剪与投影变换后的数据，因此还需要执行如下一些操作来准备这些数据。

（1）在“C:\Data”目录创建 Philadelphia 文件夹。该文件夹用于存放将来要使用的数据。

（2）将下载文件“Data\PhiladelphiaBaseLayers\clippedAndProjected”文件夹中的所有数据复制到 Philadelphia 文件夹中。

（3）按照“3.4.1 使用 QGIS 裁剪与投影变换矢量数据”中介绍的方法，将下载文件“Data\PhiladelphiaBaseLayers\clipFeature”文件夹中的 city_limits.shp 数据另存到 Philadelphia 文件夹中，并将其投影设置为“WGS 84/Pseudo Mercator”。

3.5 实践 4：使用 QGIS 处理栅格数据

在这里通过将费城 30 米分辨率的数字高程模型数据（Digital Elevation Model，DEM）处理为地形底图，来介绍如何在 QGIS 中处理栅格数据。一个美观的地形底图不仅需要包括 DEM 数据，而且需要山体阴影、坡面等图层的衬托。

DEM 数据来自美国地质调查局，位于本书下载文件的“Data\PhiladelphiaElevation”文件夹中，名为 dem.tif，将其复制到“C:\Data\PhiladelphiaElevation”文件夹中。该数据已经裁剪到费城城市范围之内，也投影到了 EPSG:3857 坐标系统中。

（1）派生山体阴影图层。

打开 QGIS，将 dem.tif 加入到当前地图中。

在图层管理器中选中 dem，然后选择栅格菜单中 Terrain Analysis 的 Hillshade 菜单项，打开“山体阴影”对话框，将输出图层格式设置为 GeoTIFF，名称设置为 hillshade，其他使用默认选项，如图 3.8 所示。



图 3.8 “山体阴影”对话框

对于那些喜欢使用命令行工具的读者，也可以使用 gdaldem hillshade 命令来提取山体阴影。

（2）派生坡度图层。

在图层管理器中选中 dem，然后选择栅格菜单中 Terrain Analysis 的坡度菜单项，打开“坡度”对话框，将输出图层格式设置为 GeoTIFF，名称设置为 slope，其他使用默认选项。在 slope 图层中，每个单元格的值为 0~90，显示该单元格的坡度。

（3）为坡度图层配色。

在 PhiladelphiaElevation 文件夹中新建一个名为 sloperamp.txt 的文本文件，其内容如下：

```
0 255 255 255
90 0 0 0
```

在每行中的第一个数值代表栅格数据中单元格的值，后 3 个数值分别代表颜色的 R、G、B 这 3 个分量。

这是一非常简单的色带，当将 slope 图层设置使用该色带后，将显示为一灰度颜色渐变，坡度值越靠近 0，颜色越淡，坡度值越靠近 90，颜色越深。当与山体阴影联合使用时，在地图中就会突出显示山体的骨架与悬崖。

打开 OSGeo4W 命令行窗口，使用 cd 命令将工作路径切换到 PhiliadelphiaElevation 文件夹。然后输入并执行如下命令，将 slope 图层设置使用 sloperamp.txt 指定的色带：

gdaldem color-relief slope.tif sloperamp.txt slopeshade.tif

在上面的命令行中，使用了 gdaldem 工具，该工具用于分析与处理高程数据。其中，为栅格配色的 color-relief 命令需要 3 个参数，按顺序分别是输入栅格数据名称、定义色带的文本文件名称以及删除文件名称。

将 color-relief 命令输出 slopeshade.tif 加入到 QGIS 地图中，显示效果如图 3.9 所示。



图 3.9　配色后的坡度图

由于大部分区域地势起伏低，所以坡度值靠近 0 的栅格单元非常多，因此配色后的坡度图大部分区域为白色，而这也正是我们要达到的效果。

（4）为高程图层配色。

在 PhiladelphiaElevation 文件夹中创建一个名为 demramp.txt 的文本文件，内容如下：

```
0 46 154 88
100 251 255 128
1000 224 108 31
2000 200 55 55
3000 215 244 244
```

使用上述色带后的高程数据，费城在地势较低的地方显示为绿色，在山地显示为黄色。

在 OSGeo4W 命令行窗口中，输入并执行如下命令：

```
gdaldem color-relief dem.tif demramp.txt demcolor.tif
```

将配色后的 demcolor.tif 加入到 QGIS 地图中，显示效果如图 3.10 所示。



图 3.10　配色后的高程图

由于上述命令将那些没有值的单元格也设置了值，因此需要重新进行裁剪工作。

选择“栅格”菜单中“提取”的 Clipper 菜单项，打开“裁剪器”对话框，按照图 3.11 所示设置参数。注意掩膜图层要使用下载文件“Data\PhiladelphiaBaseLayers\clipFeature”文件夹中的 city_limits.shp 多边形数据。



图 3.11　使用裁剪器裁剪栅格图层

在“裁剪器”对话框下面的文本框中，显示了对应的 gdalwarp 命令。

（5）地图综合设置。

在 QGIS 中新建一个项目。在地图中加入裁剪后的彩色的 DEM，即 demcolorclipped.tif。然后加入山体阴影数据，即 hillshade.tif，将其透明度设置为“60%”。加入 slopeshade.tif 数据，也将其透明度设置为“60%”。最后得到图 3.12 所示的地图。



图 3.12　彩色地形图

将 hillshade.tif、slopeshade.tif 与 demcolor.tif 这 3 个文件复制到“C:\Data\Philadelphia”文件夹中，在后面的内容中会使用到这些数据。

3.6 实践 5：PostGIS 的安装与初步使用

本实践将介绍如何使用 PostGIS，将空间数据存储到空间数据库中，而不再以文件的方式存储。

3.6.1 安装 PostGIS

在安装 PostGIS 前首先必须安装 PostgreSQL，然后在安装好的 Stack Builder 中选择安装 PostGIS 组件。

（1）下载安装程序。

PostgreSQL 安装文件下载地址是“http://www.postgresql.org/download/windows/”。编写本书时，使用的 PostgreSQL 版本是 9.4.1，所带的 PostGIS 版本是 2.1。读者也可以直接使用本书下载文件 Tools 文件夹中的 postgresql-9.4.1-1-windows.exe。

（2）安装 PostgreSQL。

双击下载的文件，所有设置都使用默认设置即可，只是需要设置超级用户名 postgres 的密码。作为练习可以使用简单的密码，例如本书为了统一也将密码设置为 postgres。在真实环境中，请注意使用安全性高的密码。

（3）安装 PostGIS。

PostgreSQL 安装完成后，提示运行 Stack Builder。通过该工具安装 PostGIS。

Stack Builder 运行后，选择安装目标软件为 PostgreSQL 9.4 on port 5432。然后在安装程序选择对话框中选择 PostGIS 2.1，如图 3.13 所示。

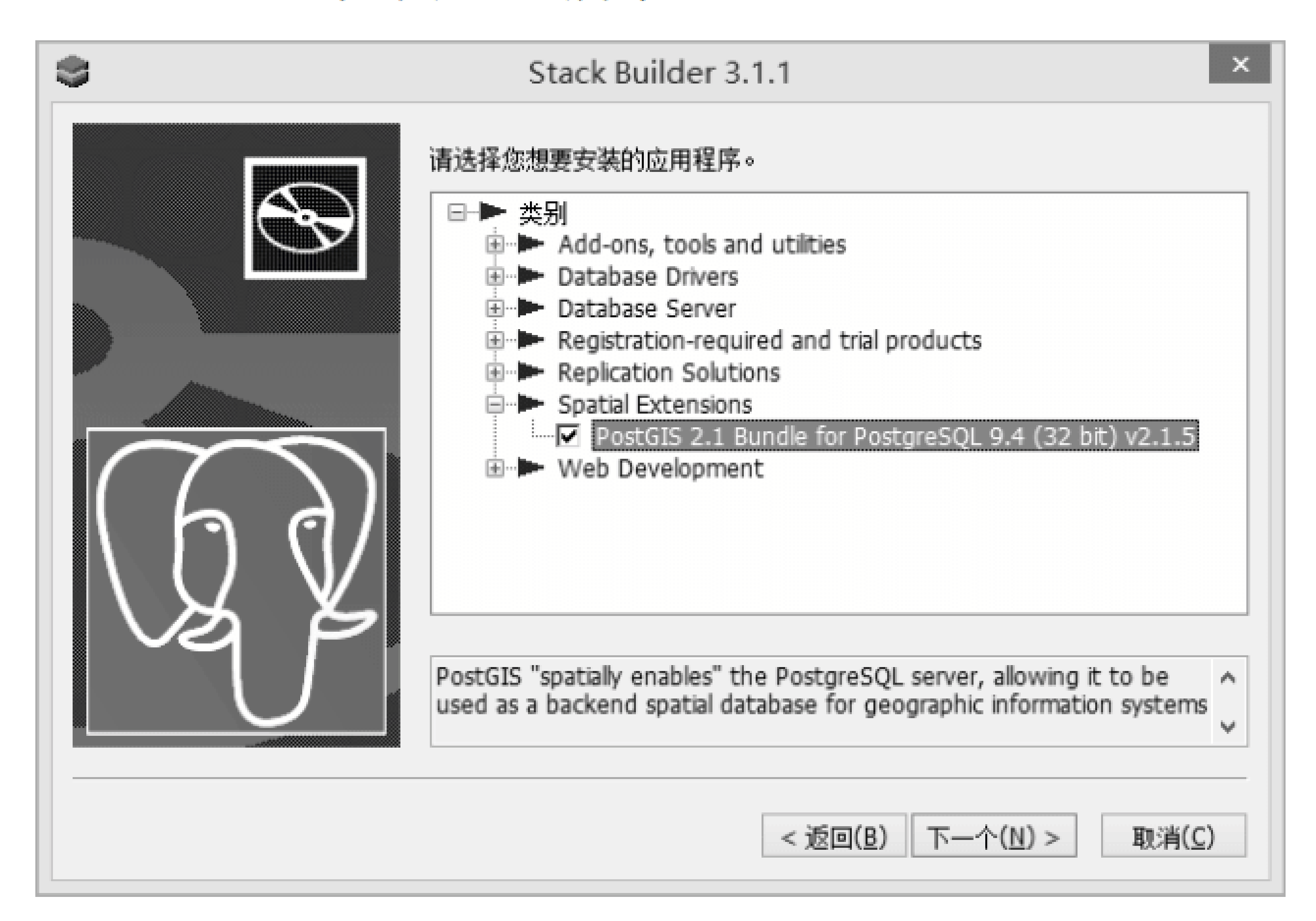


图 3.13 选择 PostGIS 2.1 作为安装程序

然后 Stack Builder 会下载 PostGIS 2.1 的安装程序。下载后就会安装，在设置安装组件时，最好选择“Create spatial database”，以便在创建数据库时可以以此作为模板，如图 3.14 所示。

对于其他步骤的设置都选择默认值即可。

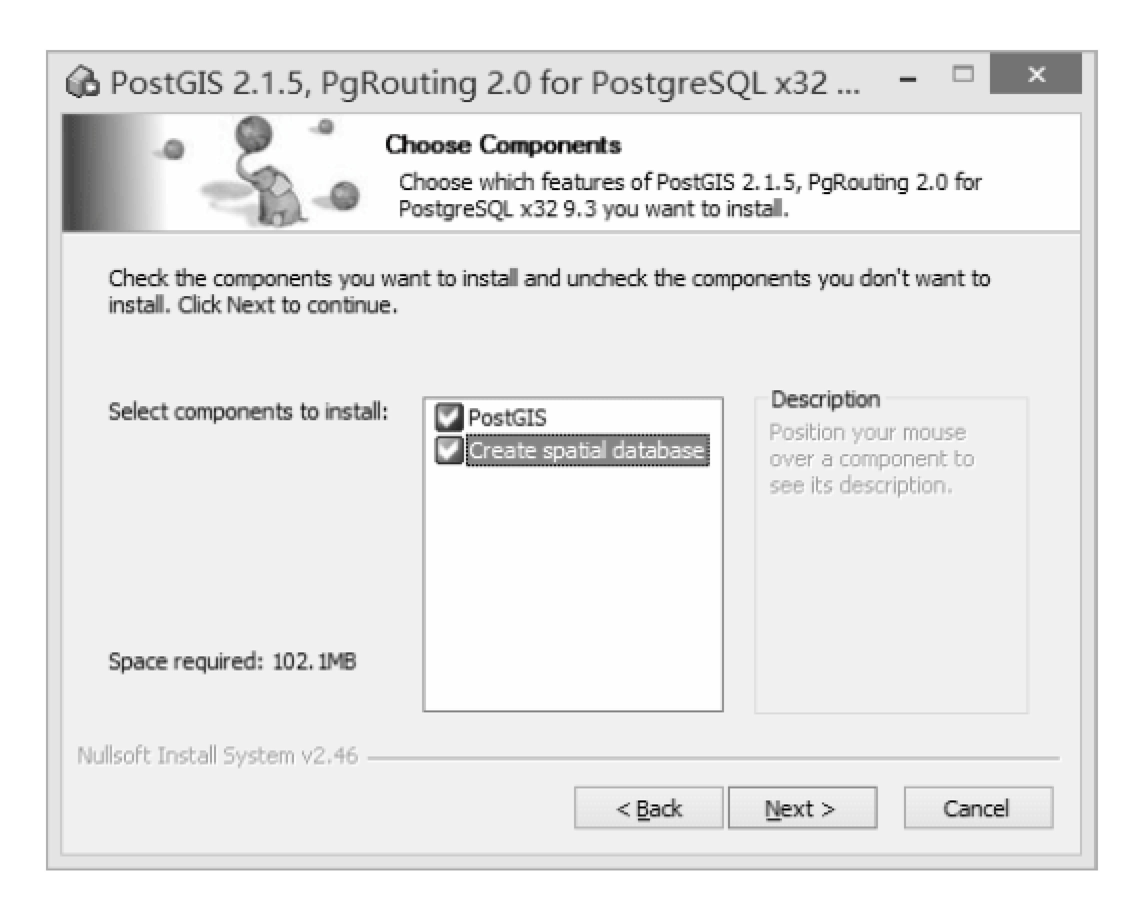


图 3.14　安装组件选择

3.6.2　创建空间数据库

要创建与管理 PostGIS 空间数据库，可使用 PostgreSQL 提供的命令行或名为 pgAdmin III 的图形化管理工具，此外像 QGIS 等 GIS 客户端软件也提供了管理插件。这里介绍如何使用 pgAdmin III 来完成创建空间数据库以及在数据库中导入空间数据。

（1）打开 pgAdmin III。

打开位于“开始>所有程序>PostgreSQL 9.4”之中的 pgAdmin III。

（2）登录到服务器。

打开 pgAdmin III 之后，发现该程序已经将本地安装的 PostgreSQL 数据库服务器列在了服务器列表中，将其选中然后选择右键菜单的“连接”命令，以超级用户 postgres 及安装过程为该用户设置的密码连接数据库服务器。连接以后，将列出该服务器中包含的内容，如图 3.15 所示，包含数据库、表空间和组角色等。

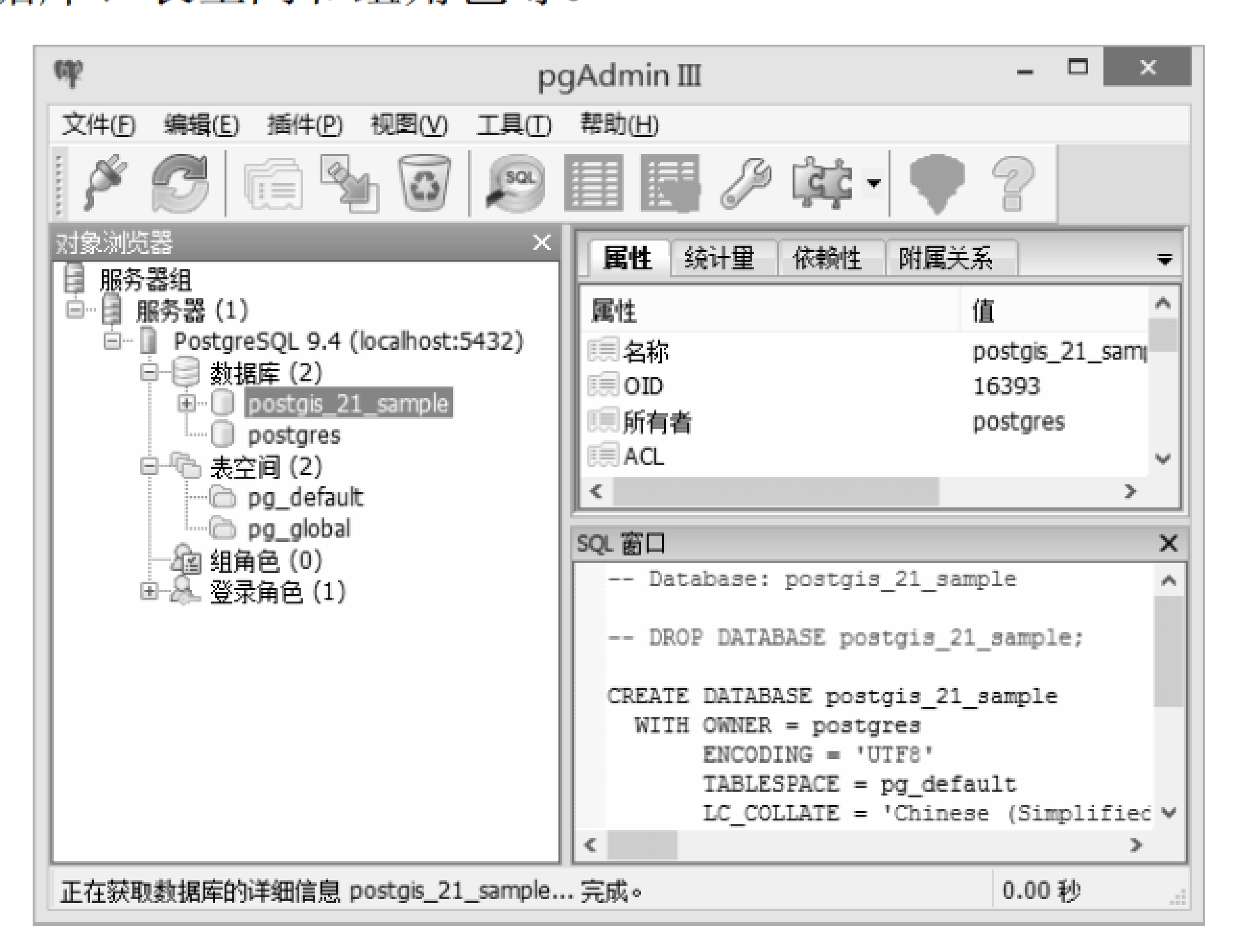


图 3.15　数据库服务器内容

（3）创建空间数据库。

选中数据库服务器内容中的数据库，然后选择其右键菜单中的“新建数据库”命令，打开“新建数据库”对话框。首先在“属性”面板中设置数据库名称，因为本书将使用加拿大温哥华市的数据，因此将数据库名称设置为“Vancouver”；并将所有者设置为“postgres”。然后，切换到“定义”面板中，将模板设置为“postgis_21_sample”。设置如图 3.16 所示。

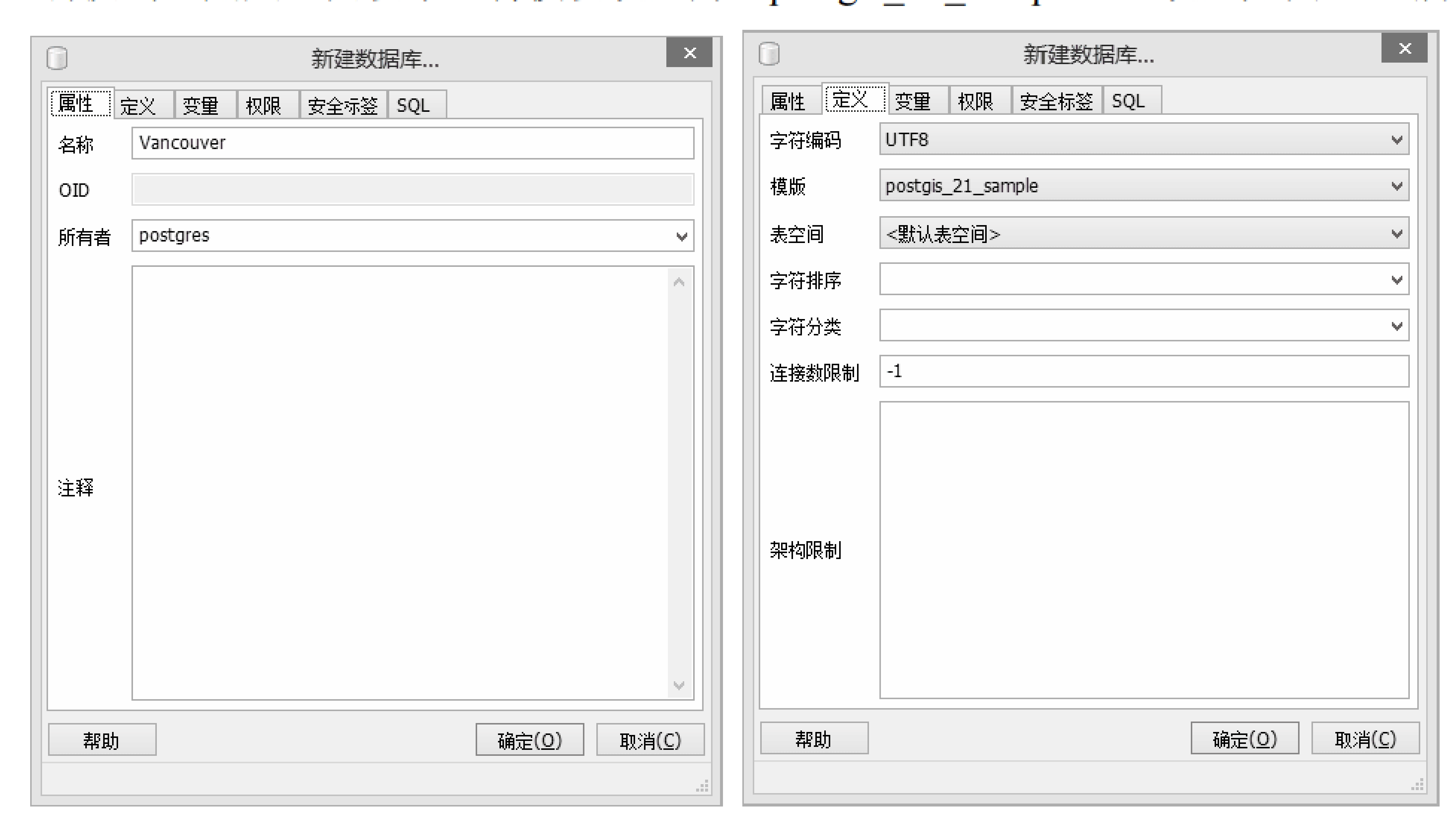


图 3.16　创建空间数据库的“属性”与“定义”面板设置

3.6.3　导入空间数据

经过前面的步骤，已经建好了功能完善的但还没有空间数据的空间数据库，接下来的工作是将空间数据导入到刚建立的数据库中。而最简单的方式是导入 Shapefile 格式的空间数据。PostGIS 提供了“PostGIS Shapefile Import/Export Manger”图形化界面工具来帮助完成 Shapefile 空间数据的导入与导出。

（1）获取数据。

本实践使用的数据位于下载文件的“Data\Vancouver”文件夹中，名为 Vancouver.shp。

（2）确定空间数据的投影系统。

在使用 PostGIS Shapefile Import/Export Manger 工具导入空间数据时，需要明确设置空间数据的 SRID，即空间引用标识符。

要确定空间数据的 SRID，有好几种方式。一种是利用“3.4 实践 3：使用 QGIS 工具裁剪与投影变换矢量数据”介绍的利用 QGIS 来确定。另一种是利用 pgAdmin III 来确定。

在“Data\Vancouver”文件夹中有一个名为 Vancouver.prj 的文本文件。.prj 文件指定了数据的投影。用文本文件工具打开该文件，可见如下一些文本：

```
PROJCS["NAD_1983_UTM_Zone_10N",GEOGCS["GCS_North_American_1983",DATUM[
```

在 pgAdmin III 中，打开查询工具，在 SQL 编辑器中输入如下 SQL 语句：

```
select srid, srtext, proj4text from spatial_ref_sys where srtext ILIKE '%NAD83 / UTM zone 10N%'
```

查询结果如图 3.17 所示，得知该空间数据的 SRID 为 26910。

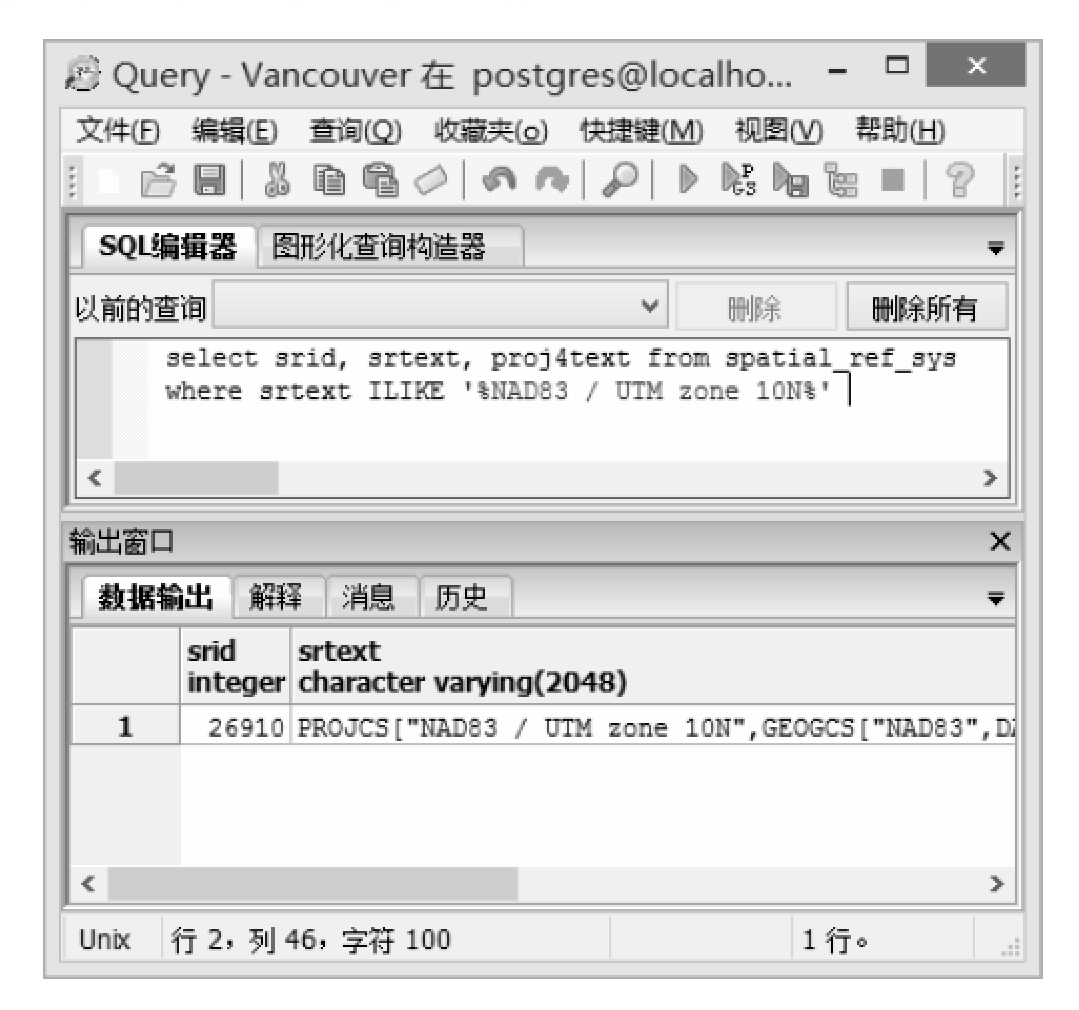


图 3.17　利用查询工具确定空间数据的 SRID

（3）导入空间数据。

打开位于“开始>所有程序>PostGIS 2.1 bundle for PostgresSQL”之中的 PostGIS Shapefile Import/Export Manger。

首先单击“View connection details”按钮，打开“PostGIS connection”对话框，输入用户名“postgres”及其对应的密码，设置连接的数据库为 Vancouver，如图 3.18 所示。

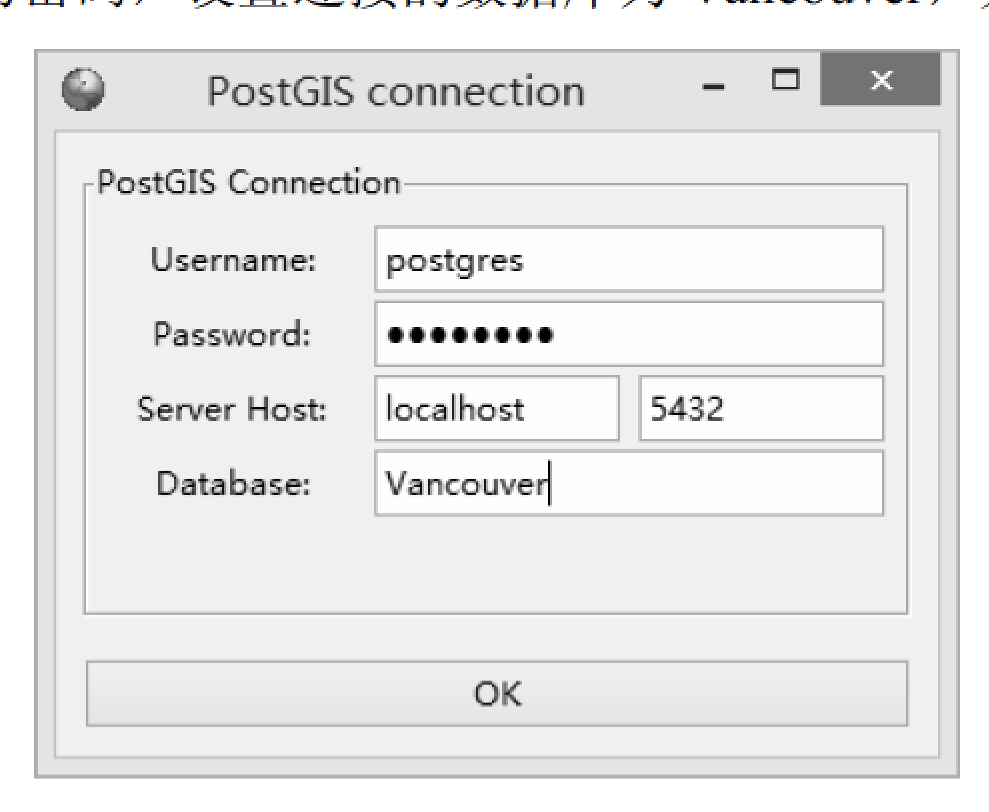


图 3.18　连接 PostGIS 数据库服务器

连接数据库之后，单击“Add file”按钮，加入 Vancouver.shp 文件，并将其 SRID 设置为“26910”，如图 3.19 所示。这一步绝对不能省略，否则不能正确导入数据。

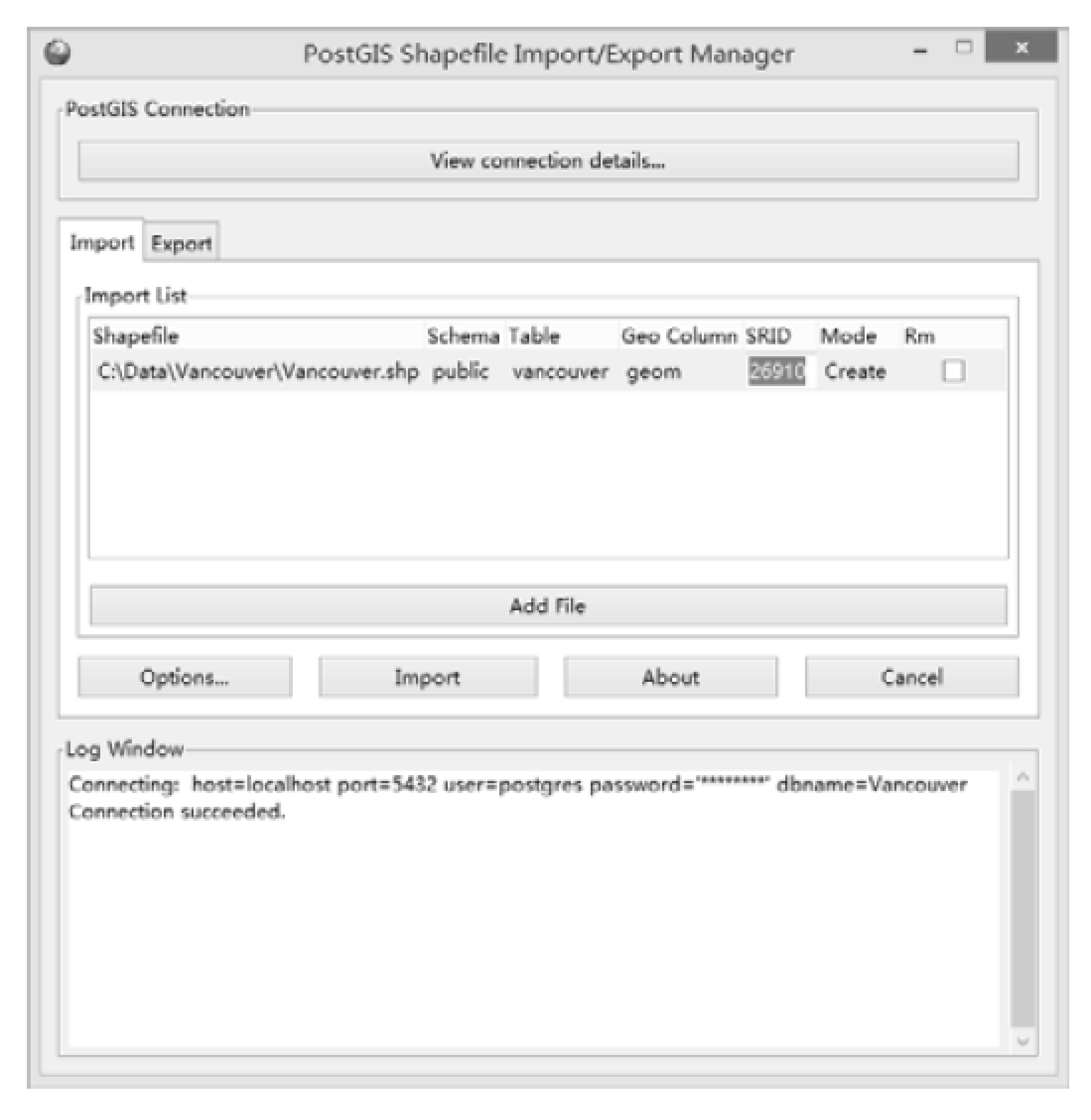


图 3.19　导入空间数据

最后单击“Import”按钮导入数据。

可以在 pgAdmin III 中通过查看 Vancouver 数据库在“架构”的 public 数据表中是否增加了 Vancouver 来判断数据是否成功导入。

（4）查看导入的空间数据。

PostGIS 并没有提供工具以地图的方式查看空间数据，不过我们可以使用 QGIS 等客户端 GIS 软件来查看。

打开 QGIS，在窗口左边的“浏览器”中选择 PostGIS，然后选择其右键菜单中的“新建连接”命令，打开“创建一个新的 PostGIS 连接”对话框，按图 3.20 设置参数，最后单击“确定”按钮连接数据库。



图 3.20　在 QGIS 中建立与 PostGIS 数据库服务器的连接

建立连接以后，便可以在“浏览器”中列出数据库服务器中所有的空间图层，如图 3.21 所示，选中某图层，将其拖入图层控制器中便可在地图中打开该空间数据。

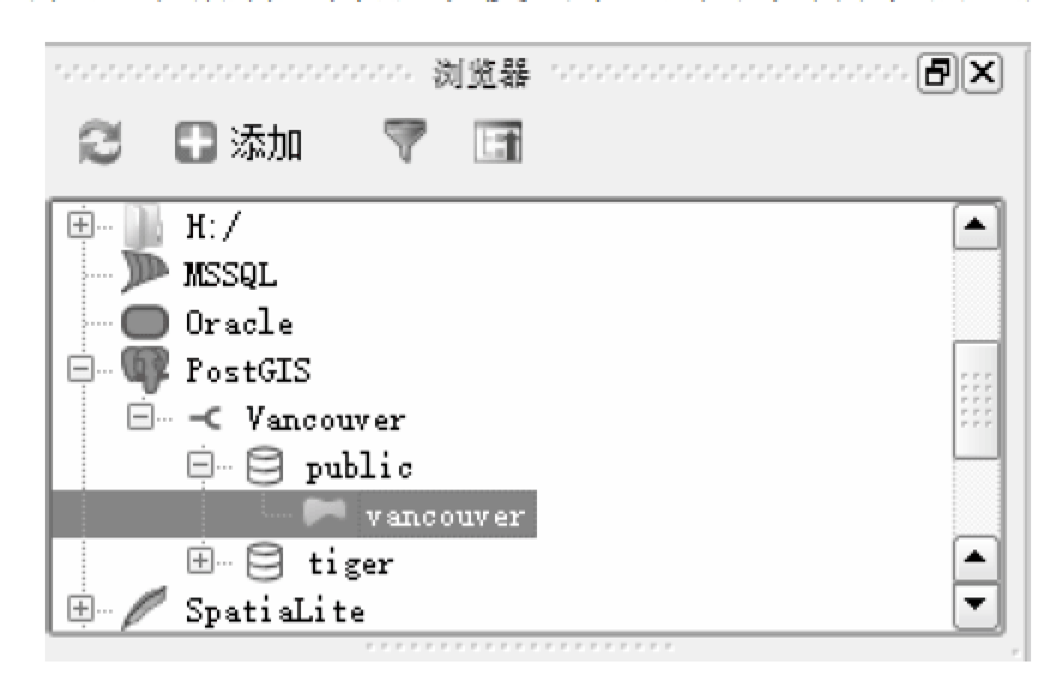


图 3.21　在 QGIS 中列出空间数据库中的所有图层

3.7　习　题

（1）GADL 与 OGR 工具的进一步学习。

访问 GDAL 工具（www.gdal.org/gdal_utilities.html）与 OGR 工具（www.gdal.org/ogr_utilities.html）介绍页面，进一步学习它们的使用方法。

（2）研究如何将栅格数据导入到 PostGIS 数据库中。

（3）从互联网上寻找一些感兴趣的数据，这些数据必须包括基础底图与专题图层。下载这些数据，将其转换为开源软件能识别的格式，并将其投影系统转换为 EPSG:3857。

第 4 章

使用 WMS 在服务器端绘制与查询地图

从本章可以学习到：

- 动态绘制地图服务
- WMS 规范基础
- WMS 的样式与符号
- 使用 GeoServer 发布 WMS 服务
- 高级符号与图层组

在前面的内容中，主要介绍如何使用 QGIS 操作本地计算机上的数据。本章将会通过 GeoServer 将地图图层发布为 Web 服务，跨入 Web 世界。

本章将着重介绍开放地理空间联盟制定的 WMS 规范。在 FOSS 领域，GIS 专业人员一直使用 WMS 来绘制 Web 地图。虽然 WMS 并没有使用最新的技术，但是却是一个被广泛使用并实用的规范，是 Web GIS 的基础。本章还将通过实践的方式，一步一步地介绍如何结合 QGIS 与 GeoServer，以及如何发布带高级符号的 WMS 服务。

4.1　动态绘制地图服务

可使用多种方法在网页浏览器中显示地图。一种是将服务器中事先绘制好的地图图像（即地图切片），发送给浏览器；第二种是从服务器上返回一串代表空间图形与属性的文本，在浏览器端绘制；第三种是在服务器端根据请求的内容绘制一个地图图像，然后返回给客户端。而后者正是本章要介绍的方式。因为每次都是根据用户请求参数，随时绘制地图，图像反映数据的最新情况，因此该方式通常称为动态地图服务。而切片地图方式只反映了生成地图切片时的数据状况。

4.1.1　动态绘制地图的优点

由于动态地图服务是在请求时访问数据并绘制的，因此对于要显示数据最新状态的需求最为有用。对于在同一时间改变位置的地理要素（例如要绘制一个大型车队中各车辆的位置），使用动态地图服务方式来绘制是最佳的选择。此外，对于那些使用切片地图来说难以生成切片、难以存储或维护的大范围地图，动态地图也是最佳的解决方案。

通过 WMS 来动态绘制地图时可以使用许多符号，还可以使用样式化图层描述符（Styled Layer Descriptors ，SLD）。如果喜欢使用 QGIS 来制图，那么还可使用其输出 SLD，并将其导入到 GeoServer 中，这样便可以让互联网的用户也可在客户端 GIS 中使用同样的样式。此外，在服务器上绘制地图可以使用很复杂的符号，通常在网页浏览器中直接绘图只能使用简单的符号。

4.1.2　动态绘制地图的缺点

等待服务器来绘制地图是一个缓慢痛苦的经历，尤其是有许多层要渲染时。对于桌面应用，2~3 秒钟的等待时间被认为是可接受，但对于“刁钻”的互联网地图用户来说，就不可接受了，因为他们既不是 GIS 相关专业，也不懂后端技术。现在，人们期待每一个地图应用像谷歌地图的响应速度一样，而这不使用地图切片是难以实现的。

如果某一个 Web GIS 有许多用户同时请求地图，那么动态地图服务便容易超负荷运行。而这就导致了两难境地：你希望地图有用，但如果服务器使用的技术是不可扩展的话，用户

越多，响应越慢。

如果知道只有有限的用户访问地图应用，例如企业内部使用的 Web GIS，使用动态绘图服务能满足用户体验要求，那么也可以不使用地图切片，从而节省了生成与维护地图切片的工作。

4.1.3 动态绘制地图的相关服务器软件

本书将详细介绍如何使用 GeoServer 来发布 WMS 提供动态地图服务。其他自由和开源的提供动态地图服务的服务器软件主要包括如下所列。

（1）QGIS Server（hub.qgis.org/projects/quantum-gis/wiki/QGIS_Server_tutorial），利用与 QGIS 桌面环境相同的函数库为用户提供 WMS 网络地图服务。只需要简单地将地图或模板的工程文件复制至服务器系统的目录下，用户就能实现桌面环境工作成果的 WMS 发布。其输出的结果与桌面环境下得到的完全一致。QGIS Server 服务器通常在 Apache 服务器环境下以 CGI/FastCGI 组件运行。

（2）MapServer（mapserver.org），由美国明尼苏达大学开发，软件建立在其他主流开源或免费系统，如 Shapelib、FreeType、Proj.4、libTIFF、Perl 等之上。最初它的开发由 NASA 支持，以使其卫星图像开放给公众。

（3）deegree（deegree.org），是一个基于 Java 的充分实现了 OGC/ISO 标准的空间 Web 服务软件，由德国在 2000 年初开始开发。

其他商业 GIS 服务器软件有 ESRI 的 ArcGIS Server 以及我国超图公司的 SuperMap Server 等。虽然 ArcGIS Server 发布的空间 Web 服务有它自己的格式，但是在发布地图服务时，也同时支持使用 WMS 规范的方式访问该服务。

4.2 WMS 规范基础

在第 1 章中介绍了 Web 服务接受用户请求，然后返回一个响应。为了保证 Web 服务的跨平台性，那么请求与响应的语法需要保证一致，而且应该有一个专门的开放规范文档加以说明。当软件开发人员创建支持该 Web 服务的服务器或客户端程序时，为了确保 Web 服务能够正确工作，他们将严格遵循规范文档。

开放地理空间联盟的 Web 地图服务（WMS）规范是一种在互联网上提供和使用动态地图时需遵守的国际规范。到目前为止，已发布了 4 个版本的 WMS 规范。这些版本是 v1.0.0、v1.1.0、v1.1.1 和 v1.3.0（最新版本）。最新版本 WMS 规范下载地址为 portal.opengeospatial.org/files/?artifact_id=14416。请读者用几分钟的时间浏览一下该文档，了解规范的基本结构，重点注意第 7 部分。

由于规范需要描述 Web 服务的每个方法及其参数，因此规范文档一般都会比较长而且比较复杂。不过幸运的是，我们并不需要直接使用规范，通常是调用那些用户友好的服务器与

客户端程序，它们屏蔽了 Web 服务的复杂性。例如，假设需要使用 GeoServer 创建一个 WMS 服务并在 QGIS 的地图中显示该服务，那么并不需要我们去参考 WMS 规范，而是直接使用这些程序提供的封装好的功能。不过可以确定的是，GeoServer 和 QGIS 的开发人员编写自己的源代码时必须详细阅读 WMS 规范。

WMS 服务主要支持以下操作：

（1）请求服务的元数据（GetCapabilities）；

（2）请求地图图像（GetMap）

（3）请求关于地图要素的信息（GetFeatureInfo，可选）。

作为基本 WMS 服务，必须至少支持 GetCapabilities 和 GetMap 操作，如果作为可查询 WMS，则需要支持可选的 GetFeatureInfo 操作。对于样式化图层描述符 WMS 服务，还有两种可选的操作，一个是请求图例符号操作，即 GetLegendGraphic；第二个是请求用户定义的样式 操作，即 GetStyles。

4.2.1　使用 GetCapabilities 操作请求服务元数据

GetCapabilities 操作返回服务的元数据。根据该服务的元数据来确定该服务支持哪些其他操作。例如在浏览器地址栏中输入如下地址：

http://eusoils.jrc.ec.europa.eu/wrb/wms_Threats.asp?&SERVICE=WMS&REQUEST=GetCapabilities

那么该地址就使用 GetCapabilities 操作访问欧洲土壤数据中心 WMS 服务的元数据。其中 http://eusoils.jrc.ec.europa.eu/wrb/wms_Threats.asp 是请求的路径，后面是参数。其中 REQUEST =GetCapabilities 表示请求的是 GetCapabilities 操作，SERVICE =WMS 表示服务是 WMS，这些都是 OGC 规范中明确要求的。

上面的地址返回或打开一个 XML 格式的文件，内容如下（为了节省篇幅，删减了一些重复与不重要的内容）：

```
<WMS_Capabilities version="1.3.0">
<Service>
  <Name>WMS</Name>
  <Title>Soil Threats</Title>
  <Abstract>Soil threats, organic Carbon Decline, Soil Erosion, Compaction, Salinization, pH </Abstract>
  <KeywordList>
    <Keyword>European soil data</Keyword>
    <Keyword> map viewer</Keyword>
    <Keyword> soil threats</Keyword>
  </KeywordList>
  <OnlineResource xlink:href="http://eusoils.jrc.ec.europa.eu/WRB/WMS_threats.asp?"/>
  <MaxWidth>2048</MaxWidth>
  <MaxHeight>2048</MaxHeight>
```

```
    </Service>
    <Capability>
      <Request>
        <GetCapabilities>
            <Format>text/xml</Format>
        </GetCapabilities>
        <GetMap>
          <Format>image/png</Format>
          <Format>image/gif</Format>
          <Format>image/png; mode=8bit</Format>
          <DCPType>
          <HTTP>
            <Get>
              <OnlineResource xlink:href="http://eusoils.jrc.ec.europa.eu/WRB/WMS_threats.asp?"/>
            </Get>
            <Post>
              <OnlineResource xlink:href="http://eusoils.jrc.ec.europa.eu/WRB/WMS_threats.asp?"/>
            </Post>
         </HTTP>
        </DCPType>
      </GetMap>
      <GetFeatureInfo>
        <Format>text/plain</Format>
      </GetFeatureInfo>
      <sld:DescribeLayer>
         <Format>text/xml</Format>
         </sld:DescribeLayer>
      <sld:GetLegendGraphic>
         <Format>image/png</Format>
         <Format>image/gif</Format>
      </sld:GetLegendGraphic>
      <ms:GetStyles>
          <Format>text/xml</Format>
        </ms:GetStyles>
    </Request>
    <Exception>
    </Exception>
    <sld:UserDefinedSymbolization SupportSLD="1" UserLayer="0" UserStyle="1" RemoteWFS="0"
InlineFeature="0" RemoteWCS="0"/>
    <Layer>
        <Name>MS</Name>
        <Title>Soil Threats</Title>
```

```
        <CRS>EPSG:3035</CRS>
        <CRS>EPSG:4326</CRS>
        <EX_GeographicBoundingBox>
            <westBoundLongitude>-28.7561</westBoundLongitude>
            <eastBoundLongitude>46.2978</eastBoundLongitude>
            <southBoundLatitude>34.1372</southBoundLatitude>
            <northBoundLatitude>68.1225</northBoundLatitude>
        </EX_GeographicBoundingBox>
        <BoundingBox   CRS="EPSG:3035"   minx="1.47e+006"   miny="2.48837e+006"   maxx="5e+006"
maxy="6.01163e+006"/>
        <Layer queryable="1" opaque="0" cascaded="0">
          <Name>OCTOP80</Name>
          <Title>Organic carbon content</Title>
          <KeywordList>
            <Keyword>Organic Carbon </Keyword>
            <Keyword> Soil </Keyword>
            <Keyword> soil organic matter</Keyword>
          </KeywordList>
          <CRS>EPSG:4326</CRS>
          <CRS>EPSG:3035</CRS>
          <EX_GeographicBoundingBox>
            <westBoundLongitude>-41.9108</westBoundLongitude>
            <eastBoundLongitude>58.1997</eastBoundLongitude>
            <southBoundLatitude>32.2238</southBoundLatitude>
            <northBoundLatitude>71.7535</northBoundLatitude>
          </EX_GeographicBoundingBox>
            <BoundingBox   CRS="EPSG:4326"   minx="32.2238"   miny="-41.9108"   maxx="71.7535"
maxy="58.1997" resx="1000" resy="1000"/>
           <BoundingBox  CRS="EPSG:3035"  minx="1.4e+006"  miny="1.98837e+006"  maxx="5.4e+006"
maxy="6.41163e+006" resx="1000" resy="1000"/>
          <Style>
            <Name>default</Name>
            <Title>default</Title>
            <LegendURL width="101" height="133">
              <Format>image/png</Format>
                <OnlineResource     xlink:type="simple"     xlink:href="http://eusoils.jrc.ec.europa.eu/WRB/
WMS_threats.asp?version=1.3.0&service=WMS&request=GetLegendGraphic&sld_version=1.1.0&layer=OCTOP8
0&format=image/png&STYLE=default"/>
          </LegendURL>
         </Style>
       </Layer>
       <Layer queryable="1" opaque="0" cascaded="0">
```

```
        <Name>PESERA</Name>
        <Title>Soil Erosion in t/ha/yr</Title>
      </Layer>
      <Layer queryable="1" opaque="0" cascaded="0">
        <Name>pH</Name>
        <Title>soil pH in Europe</Title>
        </Layer>
        <Layer queryable="1" opaque="0" cascaded="0">
          <Name>Compaction</Name>
          <Title>Natural Soil Susceptibility to Compaction</Title>
        </Layer>
          <Layer queryable="1" opaque="0" cascaded="0">
        <Name>Salinization</Name>
      <Title>Saline and Sodic Soils</Title>
    </Layer>
    </Layer>
  </Capability>
  </WMS_Capabilities>
```

其中，<Service>与</Service>之间一段的内容描述的是该服务的名称、关键词以及联系信息等。

<Capability>与</Capability>之间描述了该服务支持的操作以及包含的图层。其中<Request>与</Request>之间描述的是该服务支持的操作，从上述响应可看出该服务支持GetCapabilities、GetMap（得到地图）、GetFeatureInfo（得到地物属性）、DescribeLayer（描述图层）、GetLegendGraphic（得到图例）与 GetStyles（得到样式）操作。<Layer>与</Layer>之间罗列了该服务所包含的所有图层数据，包括 OCTOP80、PESERA、pH、Compaction 与 Salinization 这 5 个图层。

在<GetMap>与</GetMap>中的 Format 列出了 GetMap 请求所支持的返回图片的格式，包括 PNG、TIFF、GIF、JPEG、WBMP 等格式，在 DCPType 中规定了请求的方式，上面的例子表示支持 HTTP 的 Get 与 Post 两种方式。根据该响应我们可构造 GetMap 请求获取某图层或某些图层指定范围的地图。

4.2.2 使用 GetMap 操作请求地图

根据服务器的元数据，便可构造 GetMap 操作获取地图。例如要得到上述欧洲土壤数据中心有机碳含量百分比 OCTOP80 图层数据，可在浏览器地址栏中输入如下 URL：

http://eusoils.jrc.ec.europa.eu/wrb/wms_Threats.asp?&SERVICE=WMS&VERSION=1.3.0&REQUEST=GetMap&LAYERS=OCTOP80&STYLES=&CRS=EPSG:3035&BBOX=1988372,1400000,6411627,5400000&FORMAT=image/png&WIDTH=1200&HEIGHT=900

该请求返回的结果如图 4.1 所示。



图 4.1　使用 GetMap 操作请求地图

在上述的 URL 中，SERVICE=WMS 表示使用 WMS 服务；VERSION=1.3.0 表示使用 1.3.0 版本；REQUEST=GetMap 表示执行 GetMap 操作；LAYERS=OCTOP80 表示请求图层为 OCTOP80；由于没有设置 STYLE 参数的值，所以表示使用默认样式绘制图层；CRS=EPSG:3035 表示使用坐标参照系统为 EPSG:3035；BBOX=1988372,1400000,6411627,5400000 表示需要绘制的地图范围；FORMAT=image/png 表示返回的地图图片格式为 PNG；WIDTH 与 HEIGHT 指定返回图像的宽与高，单位为像素。

当从 WMS 请求地图时，有一些是必需的参数，必须提供，此外还有一些可选参数，如果 WMS 服务的发布者实现了，也可使用。在上述 URL 中所有的参数都是必需的，必须包含。对于 STYLE，由于它是必选参数，因此即使不设置其值，但仍需包含在 URL 中。可通过 WMS 规范文档（前面已给出下载地址）的 7.3.2 小节来查看哪些是 GetMap 请求必需或可选参数。

4.2.3　使用 GetFeatureInfo 操作请求地图要素信息

GetFeatureInfo 操作是一个可选的操作。GetFeatureInfo 操作仅仅支持可查属性（queryable）等于“1”的图层，对于其他图层客户端不能发送 GetFeatureInfo 操作请求。当 WMS 服务不支持 GetFeatureInfo 操作请求时，会返回服务异常信息。

GetFeatureInfo 操作的主要请求参数如表 4.1 所示。

表 4.1　GetFeatureInfo 操作请求主要参数

请求参数	是否必需	描述
VERSION=version	是	请求版本号
REQUEST=GetFeatureInfo	是	请求名称
<map_request_copy>	是	GetMap 请求参数的部分副本。不包含其中的 VERSION 和 REQUEST 参数。决定查询的目标地图，即在哪个地图图片上查询

（续表）

请求参数	是否必需	描述
QUERY_LAYERS=layer_list	是	待查询的图层列表，图层之间以英文逗号分隔
INFO_FORMAT=output_format	是	要素信息的返回格式（MIME 类型）
FEATURE_COUNT=number	否	要返回信息的要素的数量（默认为 1）。以（I, J）为中心点，根据 GetMap 操作中的请求参数 BBOX、WIDTH 和 HEIGHT 确定初始查找范围半径，对指定的查询图层进行查找。如果查询返回结果小于用户指定的 number 值，将查找半径扩大一倍继续查找，如果查询结果数目满足用户要求返回的要素数目，返回结果，否则继续扩大半径。当查找半径达到初始搜索半径的 8 倍时，终止查询，返回查询结果，进入下一图层的查询。图层的查询顺序与待查询图层列表中的顺序一致
I=pixel_column	是	以像素表示的要素 X 坐标（最左侧为 0，向右递增）
J=pixel_row	是	以像素表示的要素 Y 坐标（最上侧为 0，向下递增）
EXCEPTIONS=exception_format	否	WMS 的异常错误报告格式（默认为 application/vnd.ogc.se_xml）

可以查询欧洲土壤中心 WMS 服务中的 OCTO80，但是由于它是栅格图层，因此不能返回任何有意义的信息。这里给出一个 GetFeatureInfo 请求，只是让大家了解如何构造该请求：

http://eusoils.jrc.ec.europa.eu/wrb/wms_Threats.asp?&SERVICE=WMS&VERSION=1.3.0&REQUEST=GetFeatureInfo&LAYERS=OCTOP80&STYLES=&CRS=EPSG:3035&BBOX=1988372,1400000,6411627,5400000&FORMAT=image/png&WIDTH=1200&HEIGHT=900&QUERY_LAYERS=OCTOP80&INFO_FORMAT=text/plain&I=600&J=700

返回的响应如图 4.2 所示。



图 4.2　利用 GetFeatureInfo 发起请求地图要素信息

4.3　WMS 的样式与符号

WMS 允许调整地图中图层使用的符号。这是通过样式化图层描述符来实现的。一个样式化图层描述符描述了符号的大小、颜色和标记。样式化图层描述符比较复杂，以至于必须有它们自己的 OGC 规范文档（www.opengeospatial.org/standards/sld）定义应该如何操作它们。

样式化图层描述符可由服务发布者或客户端来设计。要设计一个真正有用的样式化图层描述符，必须了解 WMS 服务中的图层，而这可以通过可选的 DescribeLayer 操作来实现。

一旦创建了样式化图层描述符，有几种途径来使用。最常用的是将其放置到 Web 服务器中，然后在 GetMap 请求中将 STYLE 参数指向该文件的 URL。另一种方式是在 GetMap 请求的可选 SLD_BODY 参数中直接设置对应的 XML 文本。当然，后一种方式将导致 URL 非常的长，而且需要大量的特殊字符编码或转义。

样式化图层描述符中使用的 XML 往往包含许多嵌套层次而且比较复杂。因此，从头编写样式化图层描述符基本不可能的，也没有必要。可使用一个样式化图层描述符文件，在其基础上进行调整，以满足需求。另外也可以使用 QGIS，在窗口环境中样式化图层，然后将其导出为样式化图层描述符文件。这是非常有用的，但当前有一个很大的限制，那就是 QGIS 还不支持将标签信息输出到样式化图层描述符文件。

本书将会分别介绍上述两种方法。

GeoServer 中将数据与样式信息完全分开存储。在“图层”页面中定义需要发布的数据库，在“Styles”页面中定义可访问的样式化图层描述符。需要在“编辑图层”页面的“发布”面板中设置图层与样式的连接。

4.3.1　使用 GetStyles 操作请求样式

GetStyles 是 WMS 的一个可选操作。对于欧洲土壤数据中心的 WMS 服务，可使用如下地址获取样式：

```
http://eusoils.jrc.ec.europa.eu/WRB/WMS_threats.asp?version=1.3.0&service=WMS&request=GetStyles&layers=OCTOP80,PESERA
```

但是由于都是栅格图层，所以返回的是空信息。

例如对于如下 GetStyles 请求：

```
http://sampleserver1.arcgisonline.com/ArcGIS/services/Specialty/ESRI_StateCityHighway_USA/MapServer/WMSServer?version=1.3.0&request=GetStyles&layers=0,1,2
```

其响应将返回每个图层对应的样式，如下所示：

```
<sld:StyledLayerDescriptor version="1.0.0" >
  <sld:NamedLayer>
    <sld:Name>0</sld:Name>
```

```
        <sld:NamedStyle>
            <sld:Name>default</sld:Name>
        </sld:NamedStyle>
    </sld:NamedLayer>

    <sld:NamedLayer>
        <sld:Name>1</sld:Name>
        <sld:NamedStyle>
            <sld:Name>default</sld:Name>
        </sld:NamedStyle>
    </sld:NamedLayer>

    <sld:NamedLayer>
        <sld:Name>2</sld:Name>
        <sld:NamedStyle>
            <sld:Name>default</sld:Name>
        </sld:NamedStyle>
    </sld:NamedLayer>
</sld:StyledLayerDescriptor>
```

4.3.2 使用 GetLegendGraphic 操作请求图例

GetLegendGraphic 也是 WMS 的一个可选操作，用于获取图例。例如对于欧洲土壤数据中心的 OCTOP80 图层，可构造如下 GetLegendGraphic 操作以请求其图例：

http://eusoils.jrc.ec.europa.eu/WRB/WMS_threats.asp?version=1.3.0&service=WMS&request=GetLegendGraphic&sld_version=1.1.0&layer=OCTOP80&format=image/png&STYLE=default

返回结果如图 4.3 所示。

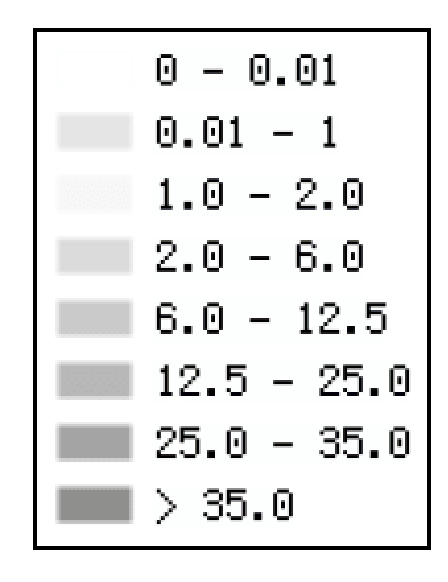


图 4.3 使用 GetLegendGraphic 操作请求图例

该图例说明了各有机碳含量百分比区间所使用的颜色，基本是颜色越深表示含量越高。只有将地图对照图例一起查看才有意义。

4.4　实践 6：使用 GeoServer 发布 WMS 服务

这里将介绍如何通过 GeoServer 将空间数据发布为 WMS 服务。

4.4.1　使用默认样式发布一个图层

（1）准备数据。

从下载文件的“Data\Neighborhoods”文件夹中，将 Neighborhoods 数据复制到费城数据文件夹中（C:\Data\Philadelphia）。该数据已经经过投影变换，投影为 EPSG:3857。

（2）启动 GeoServer。

选择“开始>所有程序> GeoServer 2.6.2 > Start GeoServer”，启动 GeoServer。

（3）进入服务器管理页面。

通过选择“开始>所有程序> GeoServer 2.6.2 > GeoServer Web Admin Page”，或者直接在浏览器中输入“http://localhost:8080/geoserver/web/”地址，进入 GeoServer 的 Web 管理页面。在 GeoServer 的 Web 管理页面中输入用户名与密码进行登录。如果是默认安装，那么用户名为“admin”，密码为“geoserver”。

（4）创建工作区。

在 GeoServer 中发布和部署地图数据涉及到的几个重要的概念——工作区（WorkSpace）、数据存储（Store）和图层组等。工作区（有时又称为命名空间）是一个用于组织类似图层数据（数据集）的容器。常常会把某个项目或工程的相关图层数据放到一个工作区里。通过工作区的使用，可以避免相同图层名的冲突。例如，在名为 beijing 工作区中的 streets 图层，引用时使用的是“beijing:streets”，这就可以与在另一个工作区中同样名为 streets 图层（dc：streets）避免冲突。而数据存储是一实际的文件夹或数据集。在一个工作区中可以包含几个数据存储，因此在引用数据存储时必须在数据存储前加上工作去的名称。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“工作区”，在右边窗口列出了 7 个示例工作区与管理工作区的两个连接，分别是添加与删除工作区。

单击“添加新的工作区”，进入新建工作区的界面，在这里需要输入工作区的名字和命名空间 URL。在 Name 文本框中输入“webgis”，在命名空间 URL 文本框中输入“http://localhost:8080/geoserver/webgis”，如图 4.4 所示。然后单击“提交”按钮。命名空间 URL 并不需要是一个真实的 URL，只需要确保它是唯一标识。



图 4.4　新建工作区

（5）在工作去中加入新的数据存储。

数据存储维护着地图数据和文件系统中的文件夹的映射关系。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“数据存储”，在右边窗口列出了 9 个示例数据存储与管理数据存储的两个连接，分别是添加与删除数据存储。

点击“添加新的数据存储”，进入新建数据源页面。在该窗口中需要确定数据源的类型。在 GeoServer 中，如果同时有栅格与矢量数据的话，则需要分别建立数据存储。在本实践中，我们使用的是矢量文件数据，因此选择“Directory of spatial files (shapefiles)”，进入新建矢量数据源窗口。

按照图 4.5 所示的方式设置各参数，将工作区设置为“webgis”，将数据源名称设置为“Philadelphia”，然后设置数据对应的文件夹。最后单击“保存”按钮。

新建矢量数据源
添加一个新的矢量数据源
Directory of spatial files (shapefiles)
Takes a directory of shapefles and exposes it as a data store
存储库的基本信息
工作区 *
webgis
数据源名称 *
philadelphia
说明
费城矢量数据
☑ 启用
连接参数
shapefiles文件的目录 *
file://C:\Data\Philadelphia 浏览...
DBF文件的字符集
UTF-8
☑ 如果缺少空间索引或者空间索引过时，重新建立空间索引
☐ 使用内存映射的缓冲区
☑ 高速缓存和重用内存映射
保存 取消

图 4.5　新建矢量数据源

要注意的是，即使在文件夹中同时还包含栅格数据，也不影响创建矢量数据源的数据存储。只是如果想使用栅格数据，那么则需要另外新建数据存储。

通过上面的设置之后，便可以指定需要发布为服务的矢量图层。

（6）发布图层。

在新建矢量数据源页面中单击“保存”按钮后，自动切换到新建图层页面。该页面列出了 Philadelphia 文件夹中所有的矢量文件。

或者，在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“图层”，在右边窗口列出了 19 个示例图层与管理图层的两个连接，分别是添加与删除资源。选择“添加新的资源”，也同样进入新建图层页面。从下拉列表框中选择 webgis:philadelphia，表示从该数据存储中选择图层。

找到 Neighborhoods 图层，然后选择“发布”连接，进入编辑图层页面。

在该页面中包含了许多发布图层的选项。在数据选项卡中定位到“坐标参照系统”部分，首先在“定义 SRS”文本框中输入“EPSG:3857”，并将“SRS 处理”设置为“强制声明”。然后通过单击“从数据中计算”与“Compute from native bounds”计算并自动填充边框坐标，如图 4.6 所示。

图 4.6　设置坐标参照系统与边框

最后在页面底部选择“保存”，进入到图层列表页面。

注意在该步骤中，如果在选择了“Compute from native bounds”之后并没有计算出经纬度表示的边框，那么可能是 GeoServer 没能正确解析“定义 SRS”文本框中输入的坐标参照系统。这时需要单击“查找”按钮，然后选择 EPSG:3857。最后再重新选择计算。

（7）预览图层。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“Layer Preview”，在右边窗口列出了发布为服务的图层与图层组。定位到 webgis:Neighborhoods 图层，然后选择 OpenLayers，将会弹出一个新的窗口，在该窗口中使用 OpenLayers 访问该图层的 WMS 服务，

如图 4.7 所示。

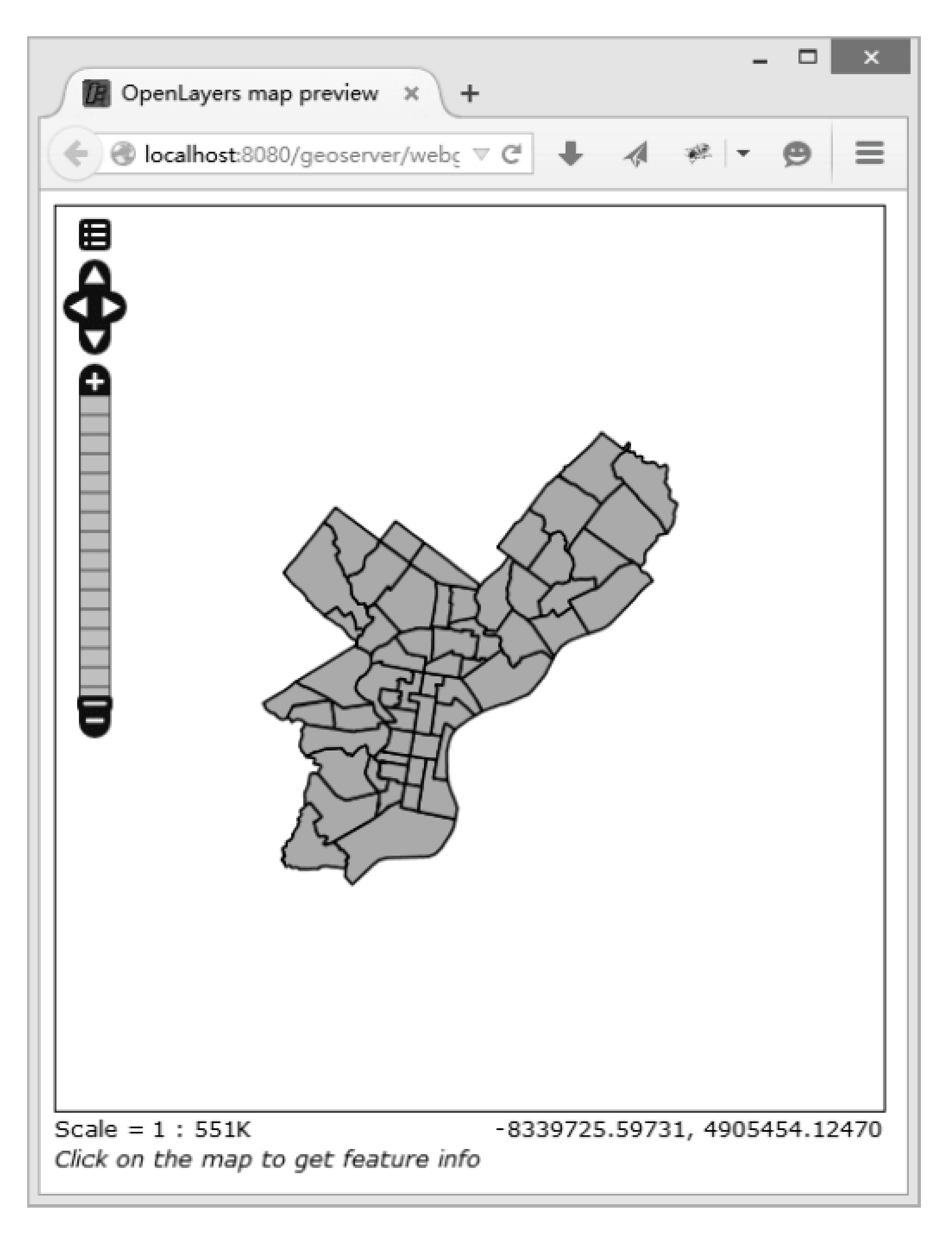


图 4.7　图层服务预览

4.4.2　使用样式化图层描述符

上述的 WMS 服务确实是包含了美国费城的社区，但是没有注记，而且也没有颜色的区分。以下将介绍如何使用样式化图层描述符，加上注记及无填充蓝色边界线条的多边形符号。这样便可将其作为专题图层叠加在其他基础底图上。

（1）下载一个带注记的多边形样式化图层描述符。

在浏览器地址栏中输入如下地址：

http://docs.geoserver.org/stable/en/user/styling/sld-cookbook/polygons.html#polygon-with-styled-label

这是一个带注记的多边形样式化图层描述符，非常接近我们的要求。我们将以此为基础来进行修改。

在上面的页面中选择"View and download the full "Polygon with styled label" SLD"，将打开该.sld 文件。

使用浏览器的另存为功能，将文件保存为 polygonwithstyledlabel.sld。

注意，不能在浏览器窗口中复制然后将其粘贴到一个文本文件中，这样做的话 GeoServer 就会不能正确识别。因为窗口中显示的内容没有包含完整的 XML 文件头。

（2）修改.sld 文件。

使用文本编辑器打开该.sld 文件，定位到 PolygonSymbolizer 与 TextSymbolizer 部分，可见到如下内容：

```
<PolygonSymbolizer>
  <Fill>
    <CssParameter name="fill">#40FF40</CssParameter>
  </Fill>
  <Stroke>
    <CssParameter name="stroke">#FFFFFF</CssParameter>
    <CssParameter name="stroke-width">2</CssParameter>
  </Stroke>
</PolygonSymbolizer>
<TextSymbolizer>
  <Label>
    <ogc:PropertyName>name</ogc:PropertyName>
  </Label>
  <Font>
    <CssParameter name="font-family">Arial</CssParameter>
    <CssParameter name="font-size">11</CssParameter>
    <CssParameter name="font-style">normal</CssParameter>
    <CssParameter name="font-weight">bold</CssParameter>
  </Font>
  <LabelPlacement>
    <PointPlacement>
      <AnchorPoint>
        <AnchorPointX>0.5</AnchorPointX>
        <AnchorPointY>0.5</AnchorPointY>
      </AnchorPoint>
    </PointPlacement>
  </LabelPlacement>
  <Fill>
    <CssParameter name="fill">#000000</CssParameter>
  </Fill>
  <VendorOption name="autoWrap">60</VendorOption>
  <VendorOption name="maxDisplacement">150</VendorOption>
</TextSymbolizer>
```

在 PolygonSymbolizer（多边形符号）部分注意 stroke（画笔）与 fill（填充）是如何设置的，在 TextSymbolizer 中注意 Font（字体）与 LabelPlacement（注记位置）的设置。还需要注意到标记为 VendorOption 的标签，表示这不是样式化图层描述符规范的内容，但是 GeoServer 支持。

下面就需要编辑该样式化图层描述符，将其修改为无填充、蓝色边界线条的多边形符号，

并佩带蓝色注记。之外还要修改的是 ogc:PropertyName 标签指定的注记字段。本示例中默认是“name”，但在 neighborhoods 矢量数据中，对于的字段名是“NAME”，需要区分大小写。

修改后的对应部分的内容如下：

```
<PolygonSymbolizer>
  <Stroke>
    <CssParameter name="stroke">#133E73</CssParameter>
    <CssParameter name="stroke-width">2</CssParameter>
  </Stroke>
</PolygonSymbolizer>
<TextSymbolizer>
  <Label>
    <ogc:PropertyName>NAME</ogc:PropertyName>
  </Label>
  <Font>
    <CssParameter name="font-family">Arial</CssParameter>
    <CssParameter name="font-size">11</CssParameter>
    <CssParameter name="font-style">normal</CssParameter>
    <CssParameter name="font-weight">bold</CssParameter>
  </Font>
  <LabelPlacement>
    <PointPlacement>
      <AnchorPoint>
        <AnchorPointX>0.5</AnchorPointX>
        <AnchorPointY>0.5</AnchorPointY>
      </AnchorPoint>
    </PointPlacement>
  </LabelPlacement>
  <Fill>
    <CssParameter name="fill">#133E73</CssParameter>
  </Fill>
  <VendorOption name="autoWrap">60</VendorOption>
  <VendorOption name="maxDisplacement">150</VendorOption>
</TextSymbolizer>
```

（3）将.sld 文件加入到服务器中。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“Styles”，在右边窗口列出了服务器中已包含的预加载样式，以及管理样式的连接。

单击“Add a new style”连接，进入 New style 页面。将新样式的名称设置为“PolygonWithStyledLabel”，将工作区设置为“webgis”。然后在页面底部浏览到样式化图层描述符 polygonwithstyledlabel.sld 文件，选择“upload”并将其上传到 GeoServer 服务器上。

如图 4.8 所示。最后单击“提交”按钮完成新建样式。



图 4.8　新建样式

在单击“提交”按钮前，还可以单击“Validate”验证是否存在错误。

（4）应用样式化图层描述符。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“图层”，在右边的页面中找到 Neighborhoods 图层，然后单击，进入编辑图层页面。

在编辑图层页面中，单击“发布”标签。定位到“WMS Settings”部分，在可获取样式列表框中选择 PolygonWithStyledLabel，然后选择向右的箭头，将其移动到选择样式中。并将默认样式设置为“PolygonWithStyledLabel”，如图 4.9 所示。最后单击“保存”按钮。

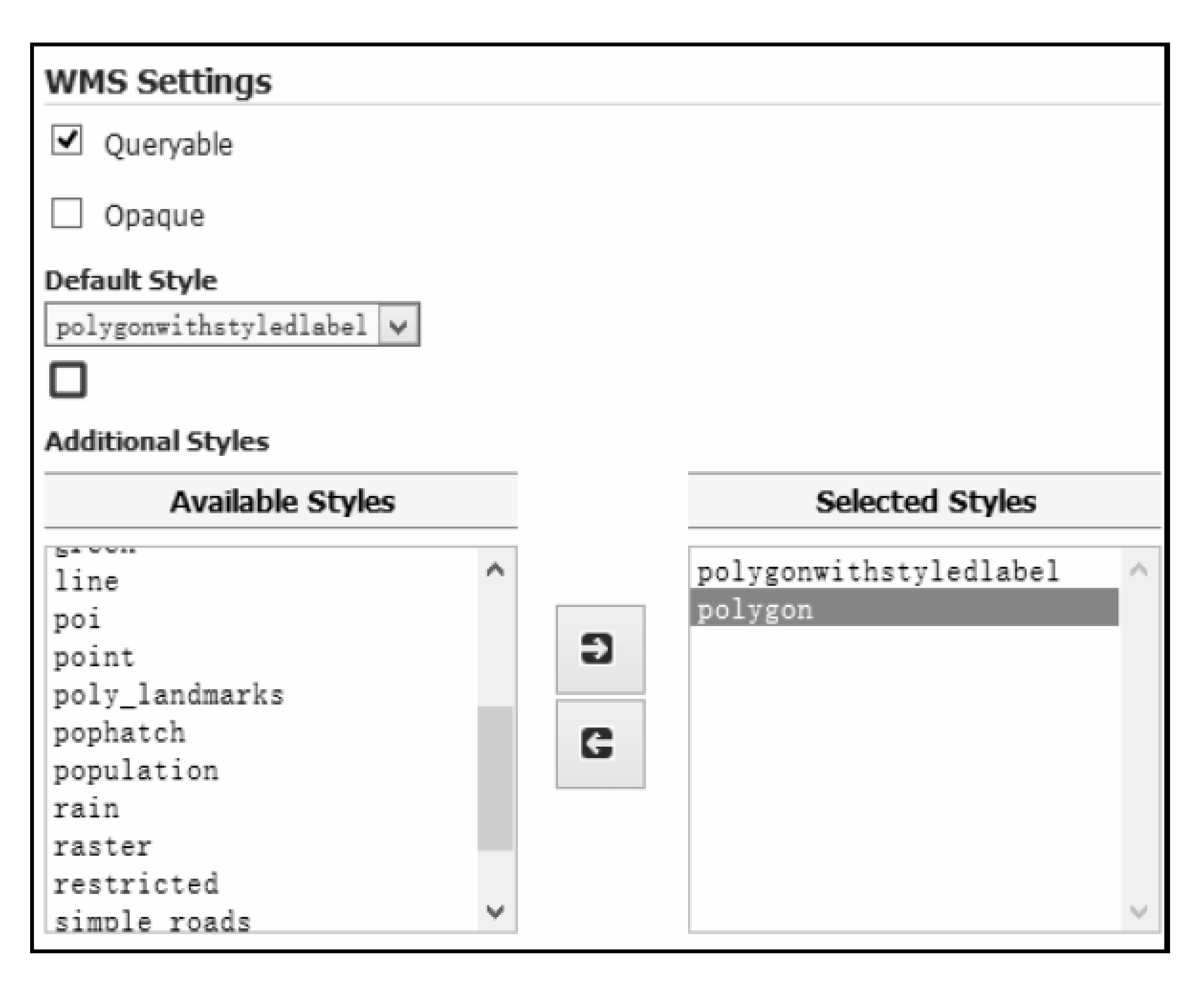


图 4.9　设置图层的样式

从图中可以看出，一个图层可以有多个样式，但是需要选择其中一个作为默认样式。对于本示例，另一样式可以为 GeoServer 自带的基本灰色多边形样式。

通过图层预览功能，Neighborhoods 图层在 OpenLayers 查看器中的显示如图 4.10 所示。

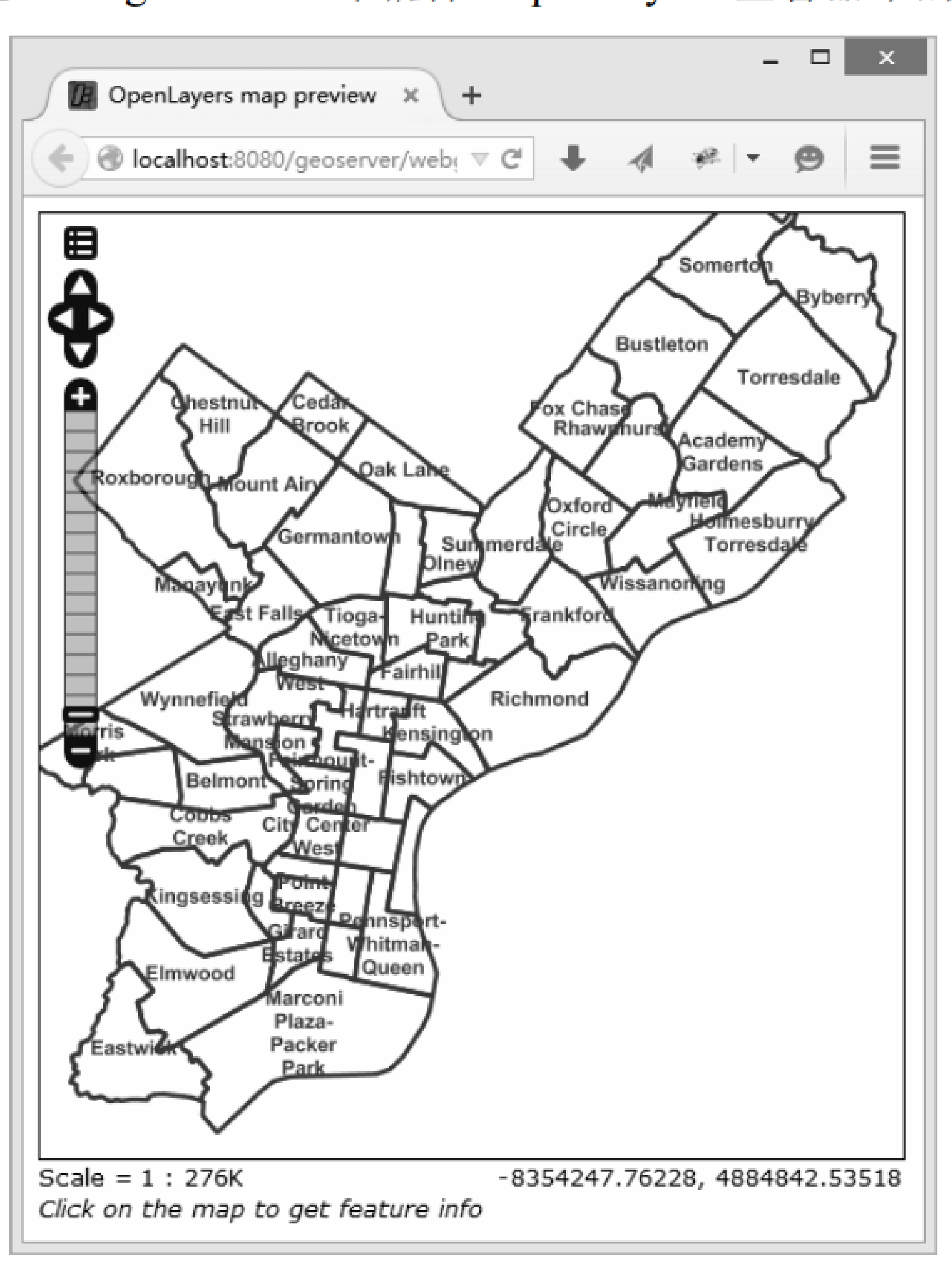


图 4.10　设置自定义样式化图层描述符后的社区图层

相对于 QGIS 或 ArcMap 等桌面程序来说，GeoServer 和 WMS 中的注记功能使用的规则

比较简单。但是注记仍然是网络地图一个非常棘手的问题。注记属于计算密集型，并且依赖于复杂的规则，必定减缓地图绘制速度。由于这些问题，Web GIS 中有时使用互动弹出窗口或文本作为替代机制。

4.4.3　在 QGIS 中访问 WMS

由于 WMS 是一标准的网络地图服务，因此很多客户端都能显示 WMS。在上面中演示了使用 OpenLayers 来预览 WMS 图层，下面将介绍如何在 QGIS 中访问 WMS 图层。

（1）新建 WMS 连接。

启动 QGIS，在窗口左边的工具条中选择“添加 WMS/WMTS 图层”按钮，打开“Add Layer(s) from a WM(T)S Server”窗口。

选择“新建”按钮，打开图 4.11 所示的“创建一个新的 WMS 连接”窗口。将名称设置为“Philadelphia Layers”，URL 设置为“http://localhost:8080/geoserver/webgis/wms?”。最后单击“确定”按钮。



图 4.11　新建 WMS 连接

（2）在地图中加入 WMS 图层。

建立连接之后，然后单击“连接”按钮，便会列出连接中包含的图层。

在图层列表中选择 Neighborhoods 图层，然后单击“更改”按钮，打开坐标参照系选择器窗口，将坐标参照系设置为 EPSG:3857。

最后单击“添加”按钮，在地图中加入 Neighborhoods 图层。

（3）在地图中叠加 OpenStreetMap 底图。

在 QGIS 中，选择“网络”菜单的“OpenLayers plugin > OpenStreetMap > OpenStreetMap”命令，将在地图中显示 OpenStreetMap 地图。

在图层控制器中，通过拖拉使 Neighborhoods 图层位于 OpenStreetMap 地图上部。

最后地图显示效果如图 4.12 所示。

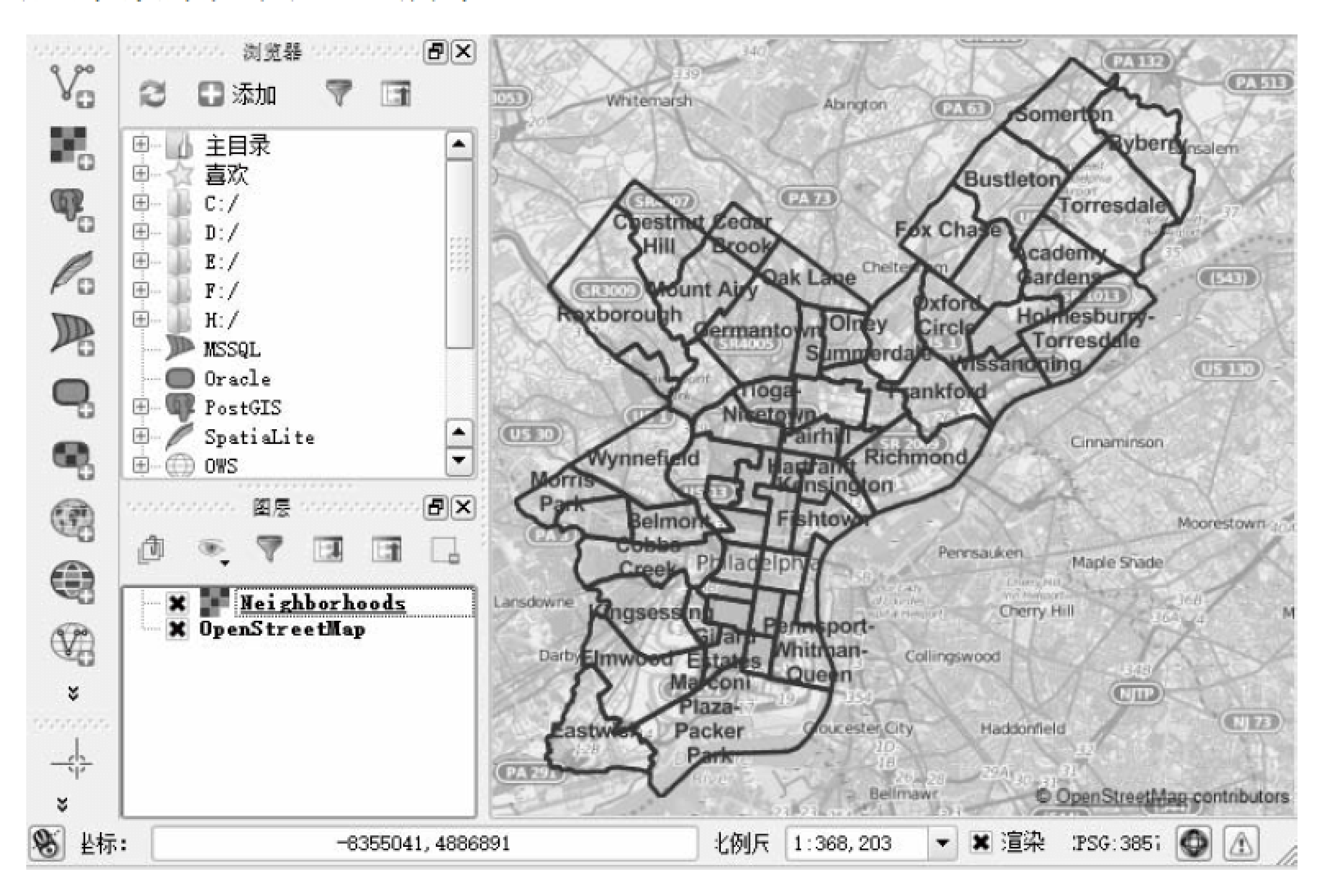


图 4.12　在 QGIS 中访问 WMS 图层

通过上面的步骤，我们创建了一个“混搭”地图，该地图引用两个不同服务器的 Web 服务。在本书的后面内容中，将介绍如何创建更实用的混搭 Web 应用。

QGIS 不仅能展示 WMS 图层，也能帮助创建样式化图层描述符。在下一个实践中，将介绍该内容。

4.5　实践 7：高级符号与图层组

在上一节中介绍了一些基本的样式化图层描述符示例，以及如何在 GeoServer 中如何样式化一个 WMS 图层。本节将介绍如何在 QGIS 中设置样式，以及在 GeoServer 中如何将多个图层组成图层组。

4.5.1　使用 QGIS 创建样式化图层描述符

QGIS 可以将当前图层的样式另存为样式化图层描述符，然后便可以使用前面介绍的方式在 GeoServer 中使用了。由于有界面化的操作，因此在 QGIS 中配置符号比直接编写 XML 要简单的多。但是要注意的是，QGIS 不能将像注记等一些样式保存到.sld 文件中。同时，由于毕竟是两个完全不同的系统，对于同样的符号，在 QGIS 与 GeoServer 中显示有稍微有些区别，因此在得到正确的结果之前可能需要一些实验，而不会一帆风顺。

（1）发布 WMS 服务。

按照实践 6 中介绍的方法，在 GeoServer 中将 Philadelphia 中的 roads.shp（道路数据）与

city_limits.shp（城市边界数据）两个文件发布为 WMS 图层。使用实践 6 创建的 webgis 工作区与 philadelphia 数据存储。

（2）在 QGIS 中配置符号。

启动 QGIS，将 Philadelphia 文件夹中的 roads.shp 与 city_limits.shp 两个文件加入到地图中。注意这里使用的矢量文件，而不是刚发布的 WMS 服务。

将 city_limits 图层的样式设置为无边线并用非常淡的灰色填充。

将道路图层的样式设置为稍微深一点的灰色细线条，线宽使用默认值。

设置后的地图如图 4.13 所示。

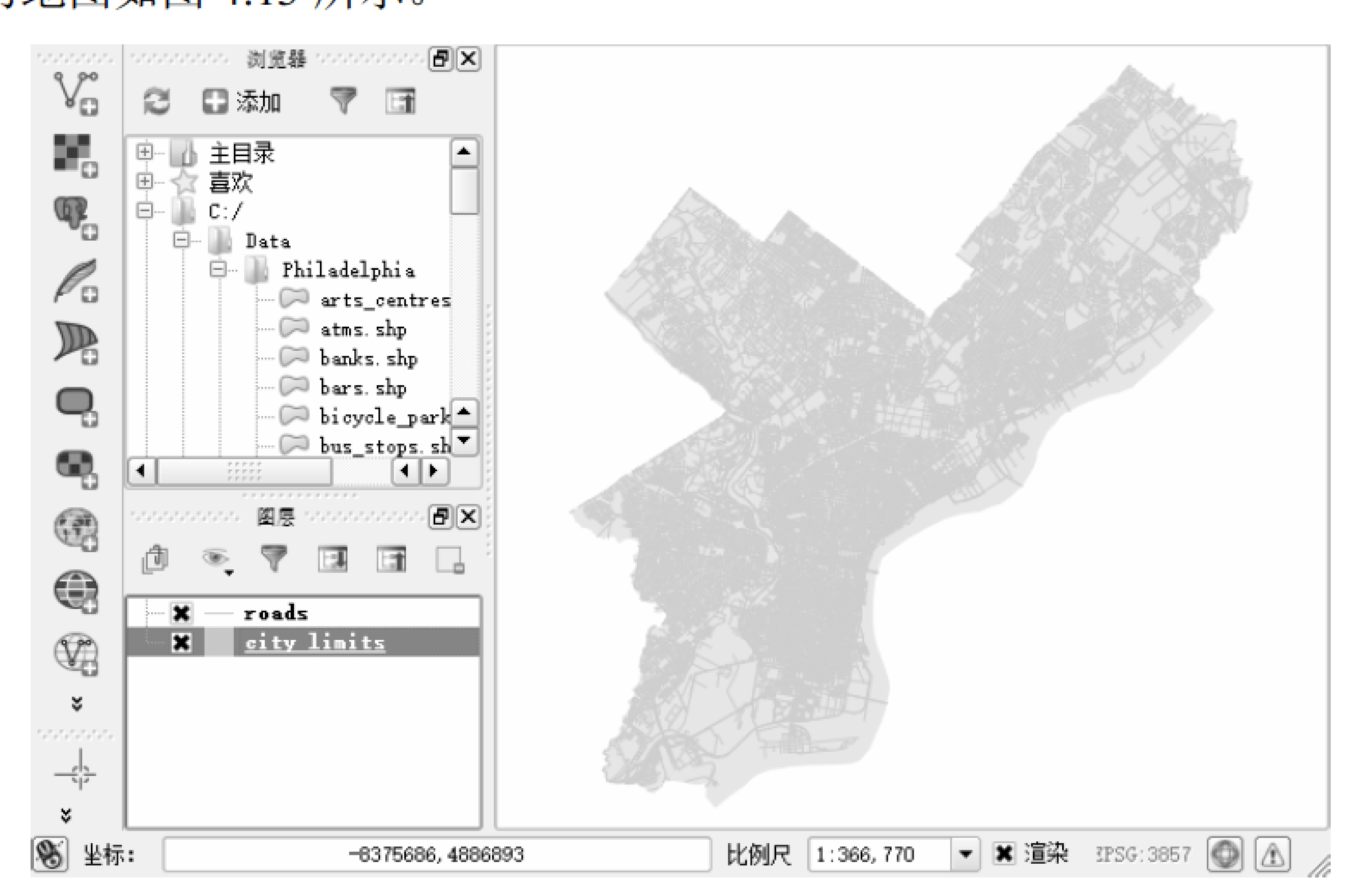


图 4.13　在 QGIS 中配置图层的样式

（3）将样式输出为 SLD 文件。

在图层列表中，双击 roads 图层，打开图层数据对话框，切换到“样式”选项卡中。选择“保存样式”中的“SLD 文件”，如图 4.14 所示，将对应的样式保存为 grayroads.sld 文件。



图 4.14　在 QGIS 中将样式保存为 SLD 文件

（4）设置 WMS 图层应用 SLD

返回到 GeoServer 的 Web 管理页面，单击左边“数据”中的“Styles”。按照实践 6 介绍的方法与步骤，在 webgis 工作区中新建一个名为 grayroads 的样式，将其样式文件指定为在 QGIS 中导出的 grayroads.sld，并将该文件上传到 GeoServer 服务器上。

按照实践 6 介绍的方法与步骤，将在 GeoServer 中发布的费城的道路图层的默认样式指定为 grayroads。并使用 OpenLayers 进行预览，确保应用了所需要的样式。

（5）设置 WMS 服务 city_limits 图层的默认样式。

重复上面的 3、4 步骤，在 QGIS 将城市边界的浅灰色样式导出为 greycitylimits.sld 文件，在 GeoServer 中创建一个使用该 SLD 文件、名为 greycitylimits 的样式，并将该样式设置为 city_limits 图层的默认样式。

4.5.2 将多图层发布为 WMS 服务

在某些情况下，需要将 WMS 作为专题图层叠加在非 WMS 底图之上。实践 6 介绍的就是这类应用。但在某些情况下，只需要一个非常简单的 Web GIS 应用，这时可以将 WMS 服务同时作为基础图层和专题图层。GeoServer 可以将多层作为一个单一的 WMS 服务进行发布。我们将使用社区图层、城市边界图层以及道路图层，将它们作为图层组进行服务的发布。

（1）先建图层组。

在 GeoServer 的 Web 管理页面窗口的左边单击“数据”中的“图层组”，在右边窗口列出了服务器中已包含的图层组以及管理图层组的连接。

（2）在图层组中添加图层。

单击“添加新图层组”连接，进入“新建图层组”页面。将“命名”设置为 NeighborhoodMap，“标题”设置为费城社区地图，“摘要”为“费城社区地图，数据来自于“Zillow.com”，“工作区”设置为 webgis。

向下滚动鼠标，定位到“图层”部分，通过“添加图层”连接，加入社区、城市边界以及道路这 3 个图层，并使用顺序箭头，按图 4.15 调整图层顺序。这里要注意的是，该列表中最上面的图层要最先绘制，第二个图层在该基础上绘制，以此类推。



图 4.15 在图层组中添加图层并调整顺序

（3）设置坐标参照系与边界。

定位到“坐标参照系”部分，将坐标参照系设置为 EPSG:3857，然后单击“生成边界”

按钮，计算服务的坐标范围并自动填写页面中“边界”相关的 4 个文本框，如图 4.16 所示。



图 4.16　设置坐标参照系与计算边界

（4）其他参数的设置。

定位到“Tile cache configuration”部分，取消其中选中的两个选项。GeoServer 可以创建切片缓存，我们将在下一章详细讲解切片地图的使用。

在页面底部选择“保存”，将返回到“图层组”页面，在列表中已经增加了我们新建的 NeighborhoodMap 图层组。

（5）预览图层组。

按照实践 6 介绍的方法，使用 OpenLayer 预览该图层组，显示结果如图 4.17 所示。

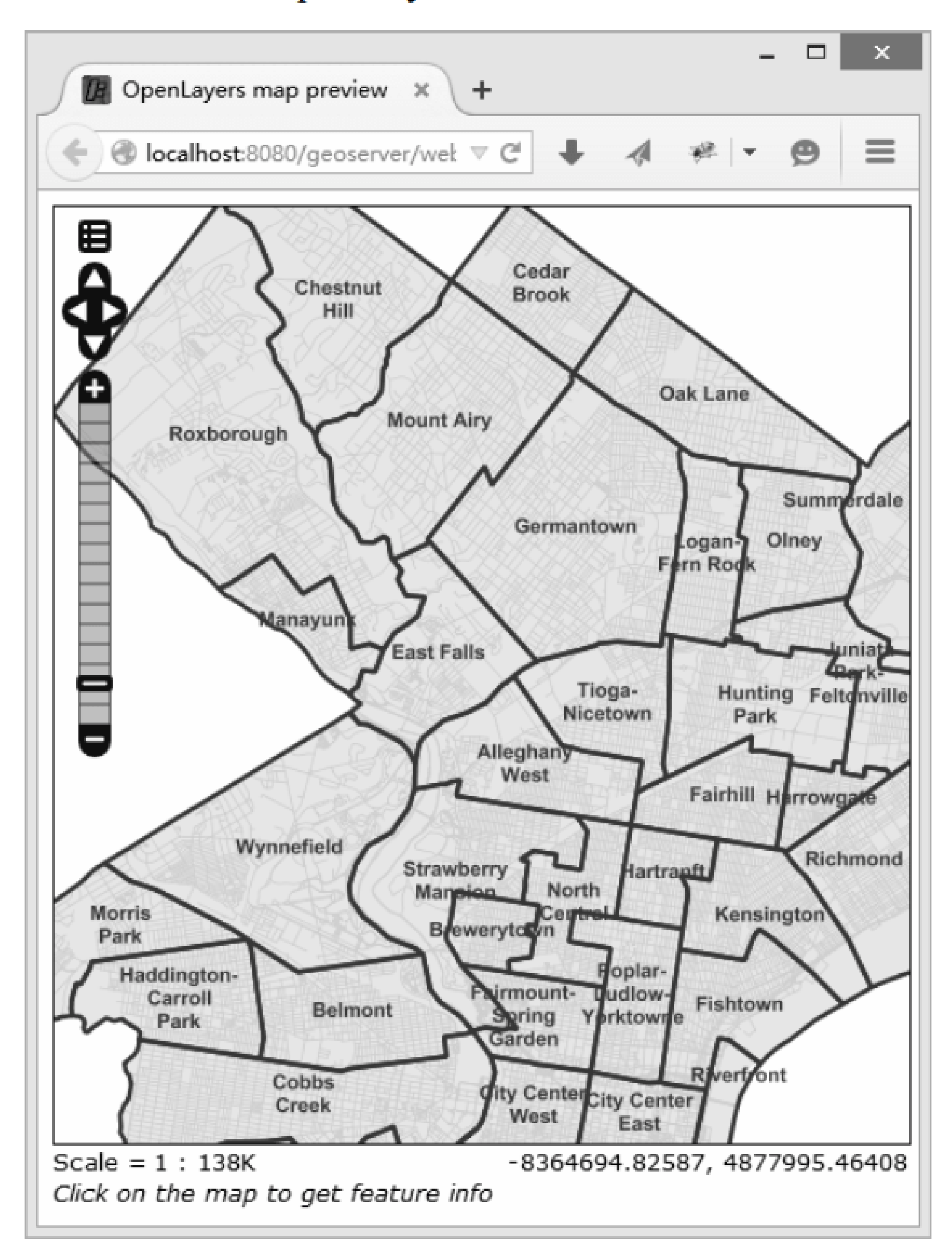


图 4.17　使用 OpenLayer 预览 NeighborhoodMap 图层组

读者可以按照实践 6 介绍的方式，在 QGIS 访问该图层组。

4.6 习 题

（1）下面是两个 WMS 服务的路径：

- http://sampleserver1.arcgisonline.com/ArcGIS/services/Specialty/ESRI_StatesCitiesRivers_USA/ MapServer/WMSServer
- http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi

编写一说明文档，描述如下内容：

- 它们都支持那些操作；
- 发布服务的组织以及它们所使用的技术；
- 每个服务包含哪些图层；
- 针对每个操作构造相应的请求，给出并分析响应结果；
- 图层样式存在哪些缺陷，如何使用本章介绍的方法加以改进。

（2）阅读如下文档，进一步了解样式化图层描述符：

- SLD 介绍（docs.geoserver.org/stable/en/user/styling/sld-introduction.html）；
- SLD 使用（docs.geoserver.org/stable/en/user/styling/sld-working.html）；
- SLD 说明书（docs.geoserver.org/stable/en/user/styling/sld-cookbook/index.html）。

（3）使用 GeoServer 以及本章介绍的方法，将第 3 章习题指定收集的数据发布为一个 WMS 服务。服务可以是一个单独的图层，也可以是一个图层组，但是必须使用 GeoServer 默认样式之外的自定义样式。编写一说明文档，包含如下内容：

- 发布服务的图层说明；
- 发布后的服务在 GeoServer 图层列表中的截屏；
- 描述制作样式的过程；
- 样式在 GeoServer 中的截屏；
- 使用 OpenLayer 预览这些 WMS 服务的截屏。

第 5 章

切片地图

从本章可以学习到：

- 为什么使用切片地图
- 何时使用地图切片
- 创建与提供切片地图服务的策略
- 使用 GeoWebCache 创建切片
- 使用 TileMill 创建切片

如果要提高 Web 地图的访问速度，使用地图切片是非常有效的方法。地图切片就是在多个比例尺下配置地图，然后提前把每个比例尺下的地图绘制为小块图片，保存在服务器上名为缓存的目录中。这样客户端在访问地图时，可以直接获取需要的小块图片拼接成整幅地图，而不是由服务器动态创建出一幅图片再送到客户端，从而极大提高了访问速度。

本章将介绍地图切片的利弊，以及创建与维护地图缓存的策略。并通过两个实践演示如何在实际工作中创建地图切片。第一个实践演示如何使用 GeoServer 的 GeoWebCache 软件来创建一个简单的缓存地图。在第二个实践中，介绍使用 TileMill 和 CartoCSS 标记语言创建比 GeoServer 更好的地图切片。

5.1 为什么使用切片地图

正如在前面内容中所介绍的，最初的 Web 地图，无论在地图中包含了多少个图层，也无论有多少访问请求，通常都是由服务器动态绘制。这也就是第 4 章介绍的使用 GeoServer 与 WMS 的方式。但是正如大家注意到的，这类地图中符号、标注与注记的选择非常有限而且难以应用。事实上，多年来为了避免妨碍性能，Web 制图者不得不使用最少的图层与简单符号来构造地图。许多情况下，在开发 Web GIS 应用时，甚至不需要专业制图人员的参与，而是由服务器管理员通过 XML 文件来定义图层顺序和符号大小等。这种情况在开放 Web 服务规范（如 WMS）与商业 Web 服务（如 ESRI 的 ArcIMS）中都存在。

造成使用这种方法的部分原因是为了使 Web GIS 应用程序看起来就像桌面系统。有时，这些应用被称为“瑞士军刀应用程序”，因为它们试图使用 Web GIS 来解决一切问题。人们希望在 Web GIS 中也能随意切换图层的可见性、重新排序图层、动态更改图层符号，以及其他所有桌面 GIS 应用程序能做的一切。讽刺的是，当这种心态盛行的时候，网络技术还远远满足不了这类需求。

在 2005 年前后，随着谷歌地图、微软虚拟地球（现在称为 Bing 地图）以及其他流行的 Web GIS 应用的出现，人们开始意识到，也许他们并不需要管理每一个图层所有属性的功能。这些互联网巨头已经开始将矢量图层融合为一张栅格化的图像，这些图像被切分为 256 像素×256 像素的图片及切片。这些图片预先生成并存储在磁盘上，以便快速分发到客户端。这样做可以同时支持成千上万个并发请求，而这对于动态地图绘制而言基本是不可能的。

正如图 5.1 显示的，切片地图采用的是金字塔模型，是一种多分辨率层次模型，从切片金字塔的底层到顶层，比例尺越来越小，分辨率越来越低，但表示的地理范围不变。切片地图通常都带有一个级别、行与列编号方案，以便将来自多个切片地图服务的切片放置到正确的位置。

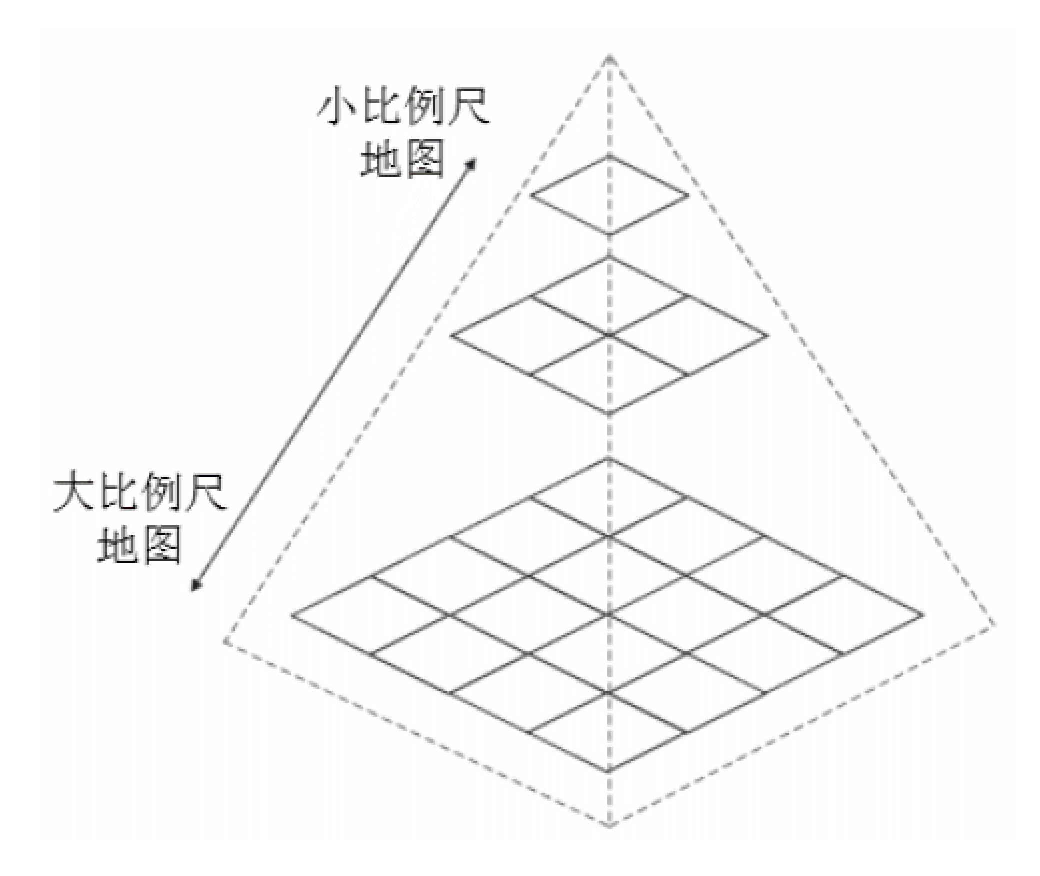


图 5.1　切片地图包含的内容

有了切片地图，制图人员再也不用担心性能问题了，他们可以使用他们所掌握的所有工具，来制作一张美观 Web 地图。一旦创建了地图切片，这些切片就保存到了服务器上的某个文件夹中，服务器检索美观地图图片与丑的图片的速度是一样的。正由于 Web 服务器可以快速分发切片地图图像，因此我们可以使用 AJAX（Asynchronous JavaScript and XML，异步 JavaScript 和 XML）编程技术来从服务器获取图片，这样当用户漫游时不会出现页面闪烁的现象。

这种变化是革命性的。一类是具有图层排序与调整符号颜色等功能，但响应非常迟缓的丑陋的地图应用；一类是没有图层控制，但具有惊人美观并且快速响应的地图应用。对于这两类 Web GIS 应用的选择，虽然对于一些 GIS 长期爱好者可能还需要停下来比较一下，但对于普通互联网用户来说根本不用思索，他们无疑会选择后者。

在谷歌地图发布了一两年以后，商业 GIS 软件开始提供创建地图切片的功能。由于可以使用成熟的地图制作工具 ArcMap，很多人选择使用 ArcGIS Server 来发布空间信息 Web 服务，但是其价格不菲。我国的超图 SuperMap iServer 是另一种商业选择。免费和开源 Mapnik 库也可以创建地图切片，但是直到最近几年才提供了将 Mapnik 封装的用户友好的应用程序（即 TileMill）。

如果一个 Web GIS 应用有成千上万用户并发访问，那么切片地图是唯一一个合理的解决方案。然而，切片地图不提供改变图层顺序与符号的功能。人们开始研究其他改进方案，将通用的基础底图图层发布为切片，在其上叠加另外的包含专题信息的图层。通用底图切片可以用于许多应用。如果专题图层的变化不频繁，或者覆盖区域非常大，则也可以使用切片方式。例如，如果使用 Firebug 等开发者工具来深入检查谷歌地图的话，可以看到其底图与专题图层（如 Panoramio 照片）都是以切片方式获取的，如图 5.2 所示。

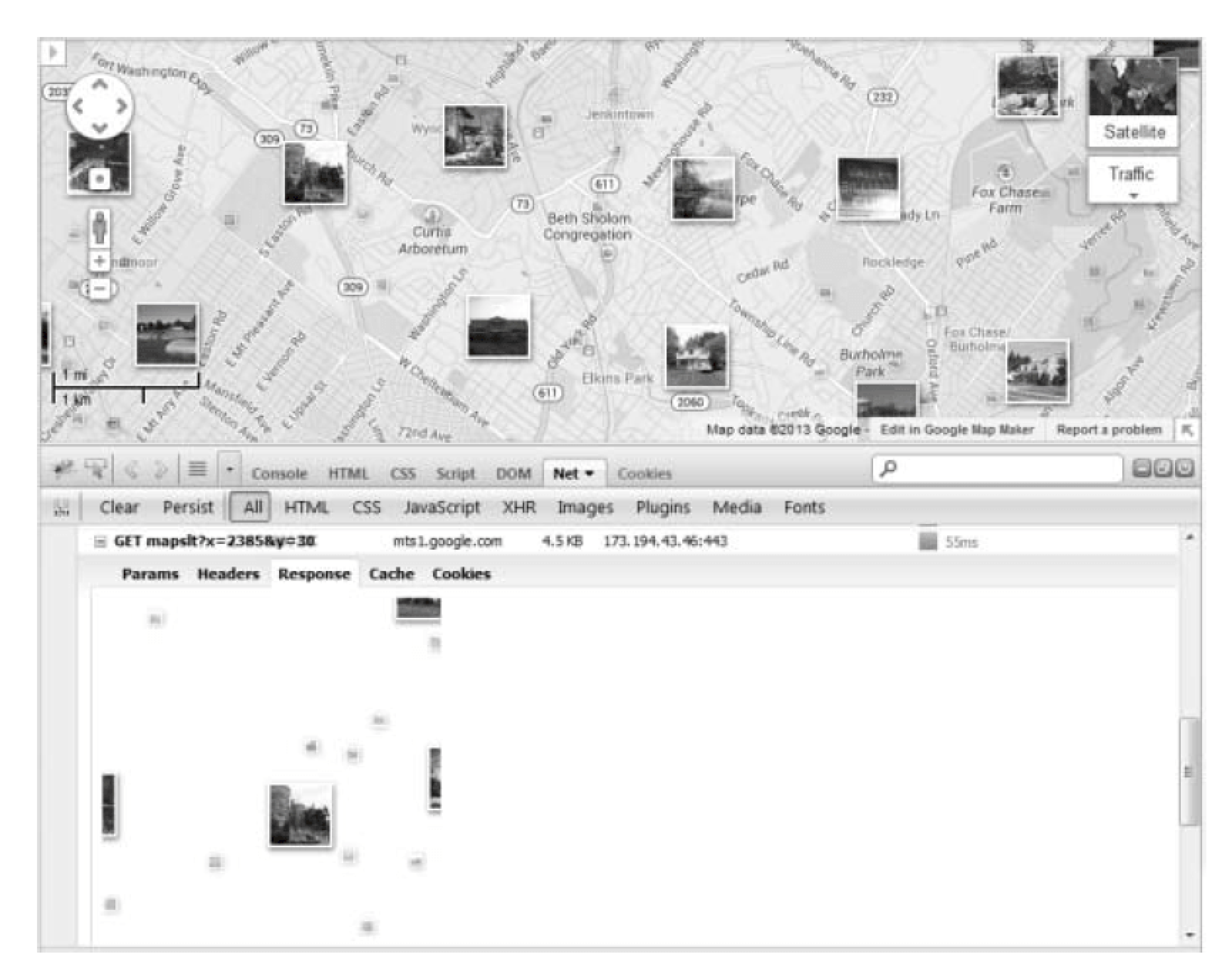


图 5.2 Panoramio 的照片专题图也以切片的方式提供

5.2 何时使用地图切片

如果希望基础底图能够快速漫游，或者同时有几十个并发用户，这时应该为底图创建切片缓存。如果专题图层中地物要素的空间位置与属性信息不经常变换，那么也可以创建切片缓存。

但不管是那种情况，必须了解到切片缓存代表着切片创建时该时刻点的地图快照。说得难听的话，当后端数据发生变化时，这些切片不会自动更新，是一些“死图片”。为了更新地图，切片地图服务提供者必须定期创建新的切片。对于那些大规模的切片缓存，有时管理员只针对变换领域更新切片，而不是重建整个地图范围的所有切片。而这又需要记录哪些地方进行过编辑，或比较几个版本的数据。

确定 Web GIS 系统是否需要创建自己的地图切片，需要考虑以下几个方面。

5.2.1 是否有满足需求的切片地图

创建切片底图需要大量丰富的数据、高端的地图制作软件和制图技能，以及潜在的大量时间和磁盘空间。正是由于存在这些挑战，通用的 Web 混搭常常使用他人创建的地图切片。如果想没有任何限制的自由与免费使用，那么 OpenStreetMap 是个最佳选择。而对于谷歌、微软或 ERSRI 的切片地图，则根据你的地图的性质（商业或不以盈利为目的），以及多少人

使用你的应用程序，确定是免费还是付费使用。而其他一些公司，例如 CloudMade 和 MapBox，则以 OpenStreetMap 数据为基础提供他们自己版本的切片。

如果决定创建自己的基础底图，那么曾经设计过多比例尺地图的有经验的制图人员是必不可少的。在每个比例尺下，地图都应该有合适的符号、颜色以及相应的详细程度。仅仅为地图所有不同的比例尺创建注记就是一项令人望而生畏的任务。此外，如果在底图中包含卫星或航空影像，那么制图人员还需要另外再单独制作一组切片，因为需要不同的颜色与符号。

5.2.2　投影

要创建切片地图可以使用任何坐标系。但是，如果准备将自己的地图切片叠加在 OpenStreetMap 或谷歌、微软、ERSRI 的切片地图上，则必须将珍贵的 GIS 数据转换到 Web 墨卡托投影。该投影创建的目的仅仅是方便将整个世界镶嵌为一组正方形切片。GIS 纯粹主义者拒绝接受该投影方式，并预测这将无法得到大规模认可，但事实与他们的期待大相径庭。

Web 墨卡托投影在很长一段时间内并没有被 EPSG 的投影数据库所接纳。EPSG 认为它不能算作科学意义上的投影，所以只是给了一个 EPSG：900913 的标号，这个标号游离在 EPSG 常规标号范围之外。因此对于一些老的软件或 API，可能使用了该代码或其他代码。直到 2008 年，EPSG 才恍然明白：不管椭球体还是球体，其实都是对地球的模拟，只是精确程度上的差别，没有本质上的不同。或者是不得不接受广泛的事实标准，EPSG 接纳了这个投影，定义投影坐标系 PROJCS 的名字为“Popular Visualization CRS / Mercator”，SRID 为 EPSG:3785；地理坐标系 GEOGCS 的名字为“Popular Visualization CRS”，SRID 为 EPSG:4055。这些标号已经进入“正常范围”。

另外要注意的是，即使使用 EPSG:3785，即 Web 墨卡托投影显示地图，也不能在该投影下进行量测功能，包括线的长度与多边形面积的量算。即使在中纬度地区，结果都存在很大的偏差。要执行量测功能，最好是将几何图形投影到本地坐标系统。

以下链接指向一个小型 Web 应用程序，该程序生动展示了 Web 墨卡托投影如何影响距离和面积计算。

http://links.esri.com/web_mercator_measurements

图 5.3 显示了如何使用示例 Web 应用程序测量一个小面的面积。在右侧面板上，有 3 个不同的坐标系用于计算此面的面积和周长。State Plane Oregon North 的测量最为准确，其次是 UTM Zone 10 测量，它们之间只相差很小的百分比。但是，请注意 Web 墨卡托投影的测量值，长度测量为 9600 米，几乎是 State Plane Oregon North 值 6763 的 1.5 倍。正如所见，Web 墨卡托投影不适合用于计算距离和面积。其他坐标系（尤其是大的面积（洲）等角投影）在测量距离和面积时同样不尽如人意。

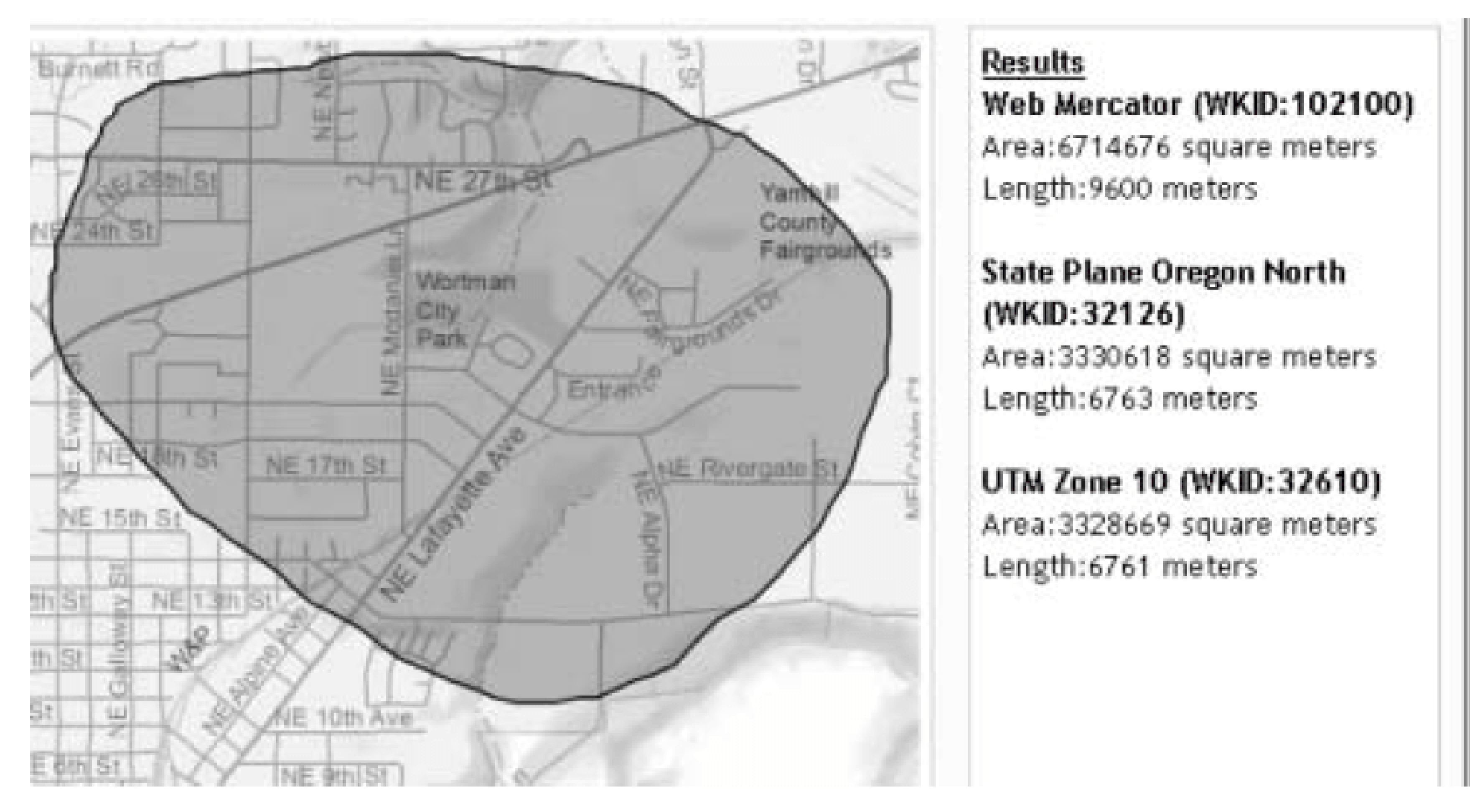


图 5.3　使用 Web 墨卡托投影时多边形的面积不准确

ESRI 的一篇名为“Measuring distances and areas when your map uses the Mercator projection”的博客文章给出了针对 ArcGIS API for JavaScript 的解决方案，使用开源软件的也可以借鉴其思路。该博客文章的地址如下：

http://blogs.esri.com/esri/arcgis/2010/03/05/measuring-distances-and-areas-when-your-map-uses-the-mercator-projection/

5.2.3　比例尺

要将地图切片叠加在 OpenStreetMap 或谷歌、微软、ERSRI 的 ArcGIS 在线切片地图上，除了保证使用相同的投影之外，还需要确保比例尺序列一致。不过这些地图的比例尺序列与我们通常见到的地形图比例尺序列很不一样，我国地形图比例尺一般有 1∶10000、1∶50000 等。而这些地图的比例尺是计算出来的，最小比例尺是将整个世界放置在 2×2 的网格中，每个网格大小是 256 像素×256 像素，因此最小比例尺为 1 : 295829355.45，再放大一个级别，那么比例尺在原基础上乘以 2，结果为 1 : 147914677.73，以此类推。由于这些比例尺很不好记忆，也很不好用于交流，并且互联网地图用户也没必要关心这么精确的数字，因此切片地图的比例尺通常简化成为“14 级”“15 级”“20 级”等。用户只需要大致了解在全国尺度、省一级尺度、城市尺度与社区尺度对应的级别即可。

表 5.1 列出了微软必应地图各比例级别对应的比例尺。该表中的比例尺与地面分辨率是以赤道位置来计算的，根据纬度不同而会有不同。

表 5.1　微软必应地图各比例级别对应的比例尺

级别	地图高与宽（像素）	地面分辨率(米/像素)	地图比例尺（96 dpi）
1	512	78271.5170	1∶295829355.45
2	1024	39135.7585	1∶147914677.73
3	2048	19567.8792	1∶73957338.86

（续表）

级别	地图高与宽（像素）	地面分辨率(米/像素)	地图比例尺（96 dpi）
4	4096	9783.9396	1∶36978669.43
5	8192	4891.9698	1∶18489334.72
6	16384	2445.9849	1∶9244667.36
7	32768	1222.9925	1∶4622333.68
8	65536	611.4962	1∶2311166.84
9	131072	305.7481	1∶1155583.42
10	262144	152.8741	1∶577791.71
11	524288	76.4370	1∶288895.85
12	1048576	38.2185	1∶144447.93
13	2097152	19.1093	1∶72223.96
14	4194304	9.5546	1∶36111.98
15	8388608	4.7773	1∶18055.99
16	16777216	2.3887	1∶9028.00
17	33554432	1.1943	1∶4514.00
18	67108864	0.5972	1∶2257.00
19	134217728	0.2986	1∶1128.50
20	268435456	0.1493	1∶564.25
21	536870912	0.0746	1∶282.12
22	1073741824	0.0373	1∶141.06
23	2147483648	0.0187	1∶70.53

5.3　创建与提供切片地图服务的策略

地图切片一般都采用简单的文件夹结构，以便提供服务。然而，由于切片数量非常多，因此它们的管理变得非常复杂。当前互联网的切片地图基本有如下两种方式组织：

（1）将切片图像以文件夹的结构放在服务器上，用户直接请求文件。在这种方法中，只需要将单独的切片图像组织在代表比例级别、行和列的文件夹结构中。很多地图 API 通过访问包含代表级别、行和列结构的 URL 来访问切片。例如，当使用 Leaflet API 访问地图切片时，必须提供格式为“http：//{s}.somedomain.com/blabla/{z}/{x}/{y}.png”的网址，其中 z 是缩放级别，x 和 y 分别是列和行。例如其中 OpenCycleMap 一个地图切片的 URL 地址如下：

http://a.tile.opencyclemap.org/cycle/10/265/420.png

（2）将切片以 Web 服务的方式提供访问。在这种方法中，虽然 Web 服务仍然需要用户提供访问切片的具体缩放级别、行与列，但是其背后文件的组织形式是看不见的。该方法比直接使用文件夹的方式相对稍微复杂一些，因此同时也会带来延时。OGC 的 Web 地图切片

服务（Web Map Tiled Service，WMTS）规范就是按照这种方式来提供切片服务的。如果使用 Firebug 或其他开发者工具来查看百度地图的话，就可以看到该模式。例如，百度地图的 URL 如下：

http://online1.map.bdimg.com/tile/?qt=tile&x=793&y=293&z=12&styles=pl&udt=20150305&scaler=1

虽然通过仔细分析，可以看到缩放级别、行与列参数，但并不能知道百度地图在后台是如何组织地图切片的。

5.3.1 创建切片地图的策略

如果地图范围覆盖广，例如一个省或国家，那么大比例尺中地图切片的数量非常巨大。然而具有讽刺意味的是，在大比例尺下，很多地图切片包含的信息却非常少。例如在 1:2250 比例尺下，居民区附近的地图切片包含了丰富的有用的信息，但是如果漫游到沙漠或海洋地区，那么切片就很可能完全是空的，没有任何有用的信息。那么我们是否还有必要花费大量的时间创建并用上千 MB 的磁盘空间来存储它们呢？

对于这种情况，我们希望能找到某种按需创建切片的软件，也就是说，在用户第一次访问该区域时创建切片。第一个漫游到该地区的用户需要等待服务器创建切片，但是接下来的用户就不需要等待了。这样一来，那些受欢迎的地区有地图切片，那些从来没人访问的地区就不需要创建与存储切片。显然，这种方法的有效性基于服务器的绘制地图切片的速度。

另一种方案是使用“没有数据”图片表明某些地区没有切片。尽管地图管理人员常常不愿这么做，但是在实际使用过程中，当用户看到该图片时，都只会责怪自己放大太多，而不会埋怨管理员为什么不提供该比例尺下的地图。

最好的方法应当是事先创建最感兴趣地区的地图切片，对于不感兴趣的区域，要么按需创建切片，要么提供“没有数据”图片。虽然作为一个地理学家，可能不太愿意把一些地方归为“不感兴趣”区域，但严酷的事实是，并不是所有的地图切片都会有均等的访问量。有研究表明，互联网地图用户的访问集中在大城市、海边和交通走廊。近期来自社交媒体的反馈，如地理微博和 Flickr 照片数据集，更能准确地揭示了地图用户最感兴趣地区。不过要说明的是，我们这里讨论的是通用用途的基础底图，而对于那些专业类型的地图，例如矿产勘查和野生动物保护，可能有截然不同的使用模式。

上述方式要求切片创建软件具有允许指定部分区域的能力。大多数软件仅仅允许指定一个矩形的子区域，但是像海边、城市等互联网地图用户所感兴趣的区域却通常不是矩形区域。因此有时需要使用一系列的矩形区域。

5.3.2 使用开源软件创建切片

当前各种 FOSS 软件中的一个基本工具就是创建网络切片地图工具。其中比较方便的是 GeoWebCache，因为它集成在 GeoServer 中。其他还有 TileCache 与 TileStache 等。

Mapnik 库是一个可为 Python 与其他语言调用的 C++编写的自由及开放源代码软件，可用于创建地图切片。Mapnik 是一个高效渲染引擎，其中包含了常用 WMS 图层中没有的高级绘图选项。虽然 Mapnik 的使用不太方便，通常需要一些 Linux 的知识以及一些实验与犯错，然而以利为目的 Mapbox 公司最近发布了一个名为 TileMill 开放源码的程序，可以在 Mac 和 Windows 上运行，以 Mapnik 为底层，提供了一个漂亮的窗口界面，从而简化了制图过程。在本章的第 2 个实践中，将介绍使用 TileMill 来创建费城的切片地图。

5.4　实践 8：使用 GeoWebCache 创建切片

如果对于在 WMS 中设置的图层与符号均感到满意，但希望提供响应速度，以及支持更多的并发用户，那么则可以考虑使用 GeoWebCache 来创建地图切片。主要是因为 GeoWebCache 完全集成在 GeoServer 中。在本实践中，将介绍如何使用 GeoWebCache 来为实践 7 中发布的 NeighborhoodMap 图层组 WMS 服务创建切片地图服务。

（1）准备工作。

启动 GeoServer，并打开 GeoServer 的 Web 管理页面。

使用 OpenLayer 预览 webgis: NeighborhoodMap 图层组，进行放大、缩小以及漫游等地图操作，注意观察性能以及地图的显示。可以观察到每次漫游时，标注都存在重新定位的现象，表明没有使用地图切片。

（2）创建地图切片。

在 GeoServer 的 Web 管理页面窗口的左边单击“Tile Caching”中的“Tile Layers”连接，在右边窗口进入 Tile Layers 页面。

在 Tile Layers 页面中，单击“webgis:NeighborhoodMap”连接，进入“图层组”页面。

在“图层组”页面中滚动鼠标，定位到“Tile cache configuration”部分。通过该部分的参数配置图层的缓存。由于是从“Tile Layers”连接进来的，因此自动选择为图层组创建缓冲切片。如果不进行进一步的操作，那么 GeoServer 将按需创建缓存切片。但是我们需要的不是按需创建，而是预先创建。

在 GeoServer 的 Web 管理页面窗口的左边，再次单击“Tile Caching”中的“Tile Layers”连接，然后在右边窗口 Tile Layers 页面中的 webgis:NeighborhoodMap 行单击“Seed/Truncate”连接，将打开一个新的窗口。

在新窗口中，按图 5.4 设置“Create a new task”表单。

然后选择底部的“Submit”，页面将进入执行任务监控页面，而 GeoServer 则在后台针对不同比例尺绘制地图。

等待大约 30 秒钟以后，单击“Refresh list”连接，可以看到图 5.5 所示的进度显示，告诉已进行多长时间，估计还需要多长时间。当单击“Refresh list”连接后，该进度统计列表消失时，表示地图切片已经创建完成。

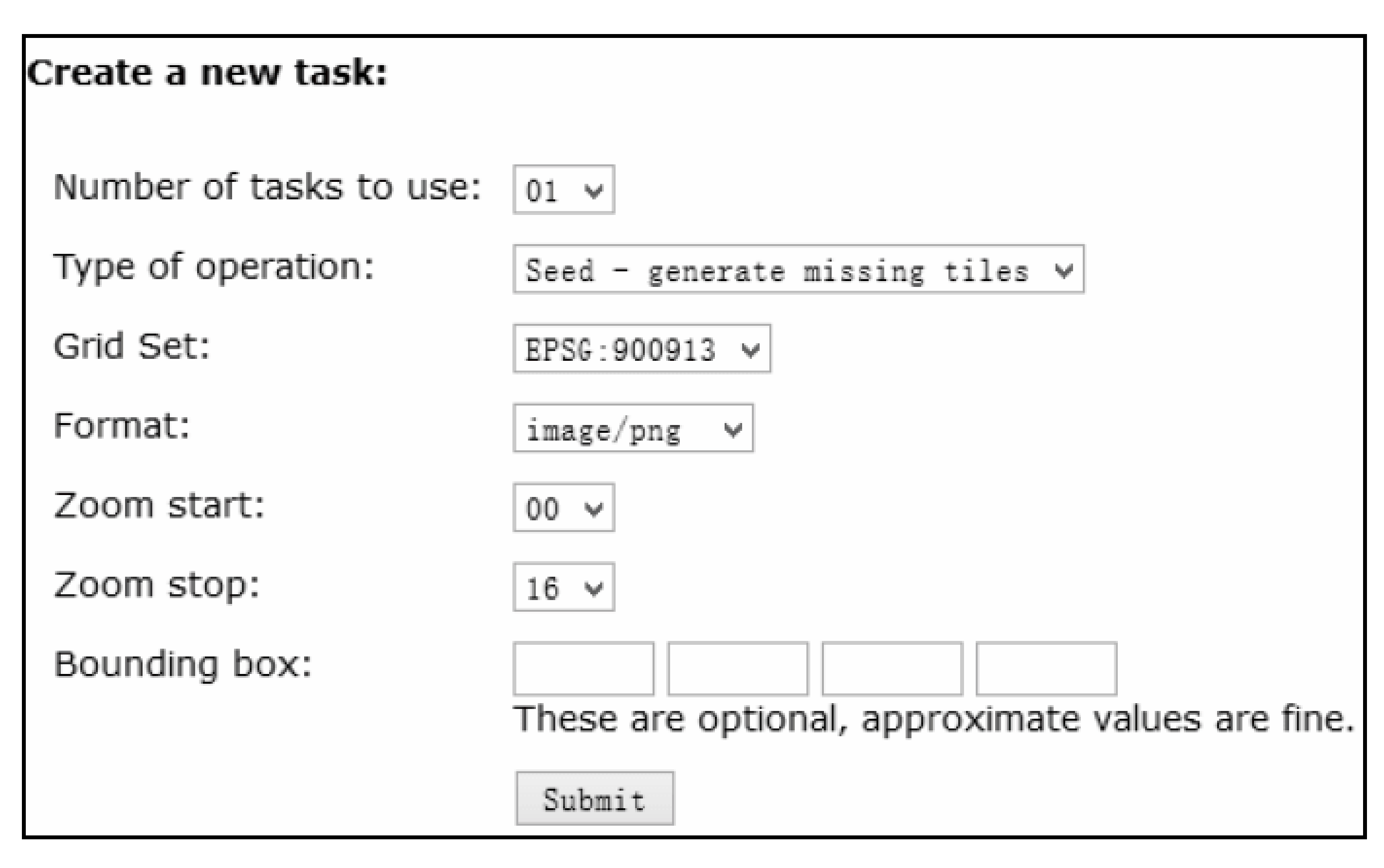
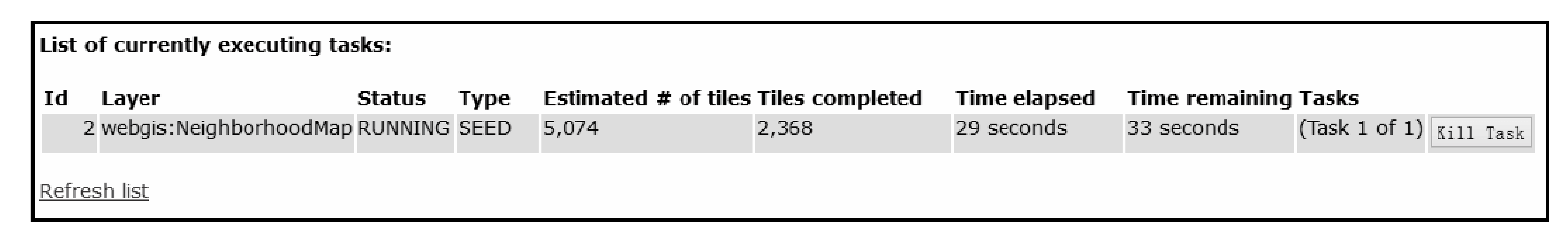


图 5.4　填写创建地图切片的参数


List of currently executing tasks:

Id	Layer	Status	Type	Estimated # of tiles	Tiles completed	Time elapsed	Time remaining	Tasks	
2	webgis:NeighborhoodMap	RUNNING	SEED	5,074	2,368	29 seconds	33 seconds	(Task 1 of 1)	Kill Task

Refresh list

图 5.5　地图切片任务执行统计列表

（3）预览切片地图。

在 GeoServer 的 Web 管理页面窗口中单击“Tile Layers”连接，进入 Tile Layers 页面。然后从 webgis:NeighborhoodMap 行的预览下拉列表框中选择“EPSG:900913 / png”，如图 5.6 所示。

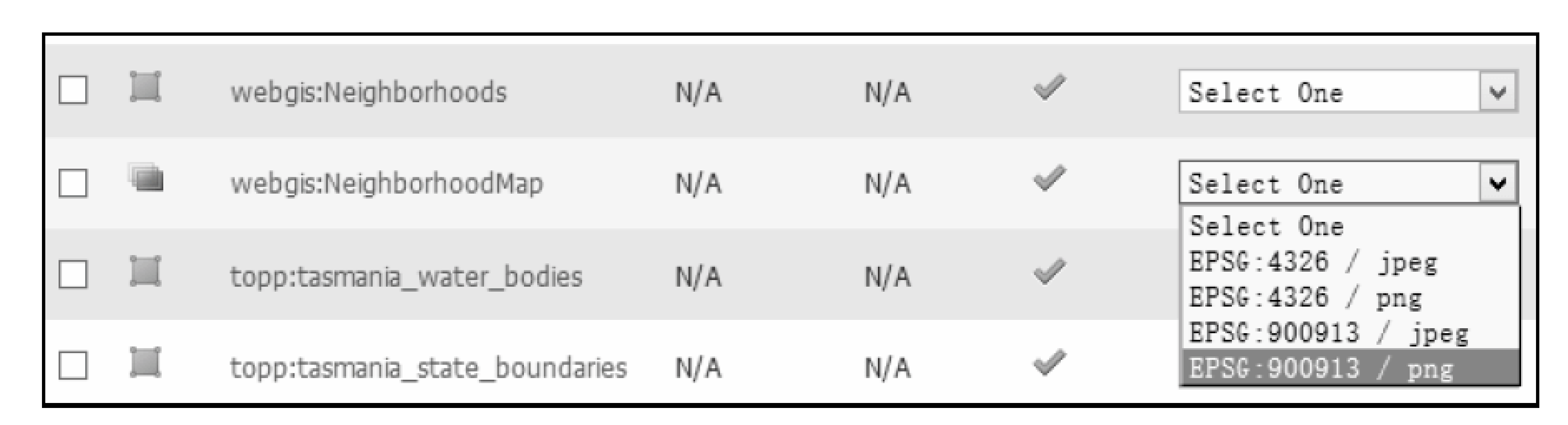


图 5.6　使用“EPSG:900913 / png”方式预览地图

在新的地图预览窗口中，对地图进行放大、缩小与漫游等操作，可以发现无须等待而地图立即显示，而且当漫游时标注并没有改变位置，表明已经利用切片缓存。

请注意，确保使用的是“Tile Layers”预览，而不是“Layer Preview”预览。切片图层预览使用的 URL 稍微有些不同，以表明需要使用切片缓存。

虽然使用切片缓存改善了性能，但是也正如大家注意到的，地图中存在重复标注的现象。因为每个地图切片并不清楚相邻切片中的标注，因此在生成地图切片时很难避免重复标注。要缓解该问题，切片生成软件通常在比切片更大的范围内绘制地图，然后再将其切开为单独的切片。GeoWebCache 将该较大区域称为“metatile（元切片）”，而 ESRI 称为“supertile（超级切片）”。可以在 GeoWebCache 调整元切片的大小进行试验，可以看到，标注重复现

象得到了很大的改善，但是仍然在元切片的边界处出现同名标注。这是因为，当对元切片进行标注时，标注放置位置引擎未识别出相邻元切片上的标注。事实上，标注引擎可能正在努力使元切片中包含尽可能多的标注，进而在边附近放置了一些标注。在相邻的元切片上可能发生同样的情况，使得元切片边界附近出现同名标注。下一个实践将介绍的 TileMill 也有元切片的概念。

5.5　实践 9：使用 TileMill 创建切片

在该实践中，将介绍如何使用 Mapnik（TileMill 是该库的封装）来创建费城通用的地图切片。本书后面内容中的相关专题图层可叠加在该基础底图上。这里所使用的数据是第 3 章中处理好的数据。如果是遵循本书给出的操作，那么数据位于“C:\Data\Philadelphia”文件夹下，使用的是 EPSG:3857 投影。

Mapnik（www.mapnik.org）是一常用的地图切片生成 FOSS 软件。Mapnik 集成了抗锯齿功能（通过混合物体边缘附近的前景像素和背景像素而使人眼看到的边界更平滑的一种图形技术）。它并不依赖任何其他 GIS 软件框架，也正因为这一特点，从而改善了性能。Mapnik 支持多种类型的文件形式与空间数据库的数据。

要使用 Mapnik，首先要定义一系列的数据源，然后将它们与一组样式规则联系起来。可使用 XML 文件定义样式规则，也可以使用 Python 等编程语言编写。一旦定义了数据源与样式规则，便可使用 Mapnik 输出地图图片。

对于初学者特别是编程技能弱的人来说，使用 Mapnik 有一定的难度。不过可以通过有图形界面的 TileMill 来帮助使用 Mapnik。在 TileMill 中，不再使用 XML 或代码来定义样式，而是使用 CartoCSS。CartoCSS 是一种语法类似 CSS（Cascading Style Sheets，层叠样式表，一种对网页进行设计的样式语言）的制图样式描述语言。如果熟悉 CSS 的话，尽管二者所包含的要素、属性等内容和含义完全不同，那么也还是会比较容易理解 CartoCSS 这种对地图进行样式设计的语言。

虽然 TileMill 是自由与开源的软件，但它是由 Mapbox 开发，并将其集成到公司的商业地图切片服务中。当使用 TileMill 输出切片时，结果是以.mbtiles 为后缀名的文件。如果不想将其发布者 Mapbox 的服务器上，可以将该.mbtiles 文件解压为一系列的切片文件，然后将其发布到自己的 Web 服务器上。

5.5.1　使用 TileMill 设计地图

（1）下载与安装 TileMill。

从 https://www.mapbox.com/tilemill/下载 TileMill，或者直接使用下载文件中 Tools 文件夹下的 TileMill-v0.10.1-Setup.exe。使用默认设置安装 TileMill。在安装 TileMill 时可以一并安装 Mapnik。

（2）在 TileMill 中创建项目。

从 Windows 的启动菜单中，选择“TileMill > Start TileMill”，启动 TileMill。

选择“New Project”创建一个新的项目，按图 5.7 设置新项目的基本设置，然后单击“Add”按钮。

图 5.7　设置新项目的信息

如果是要创建遥感影像的切片，那么为了减小切片大小，可选择 jpeg 作为切片的格式。由于这里使用的是矢量地图，因此 PNG 格式更合适。

（3）设置地图背景颜色。

在项目列表中单击 PhillyBasemap，进入地图编辑器中。此时地图预览显示在左边，右边显示的是 CartoCSS 代码。每当保存 CartoCSS 代码时，地图更新一次。

在 CartoCSS 代码窗口中，将默认的地图背景颜色修改为白色，代码如下：

```
Map {
    background-color: #FFFFFF;
}
```

单击“保存”按钮，地图预览窗口的背景颜色立即更新为白色。

接下来，就需要增加图层并设置它们的样式。

（4）增加图层。

在 TileMill 窗口最左边的工具条的最下面单击“图层”按钮，弹出图层管理器小窗口，在该窗口中单击“Add Layer”按钮。该过程如图 5.8 所示。



图 5.8　通过“图层”按钮打开图层管理器

单击“Add Layer”按钮后，弹出的是 Add Layer 窗口。在该窗口中，增加 city_limits.shp 文件作为新图层，并按图 5.9 设置其他参数。

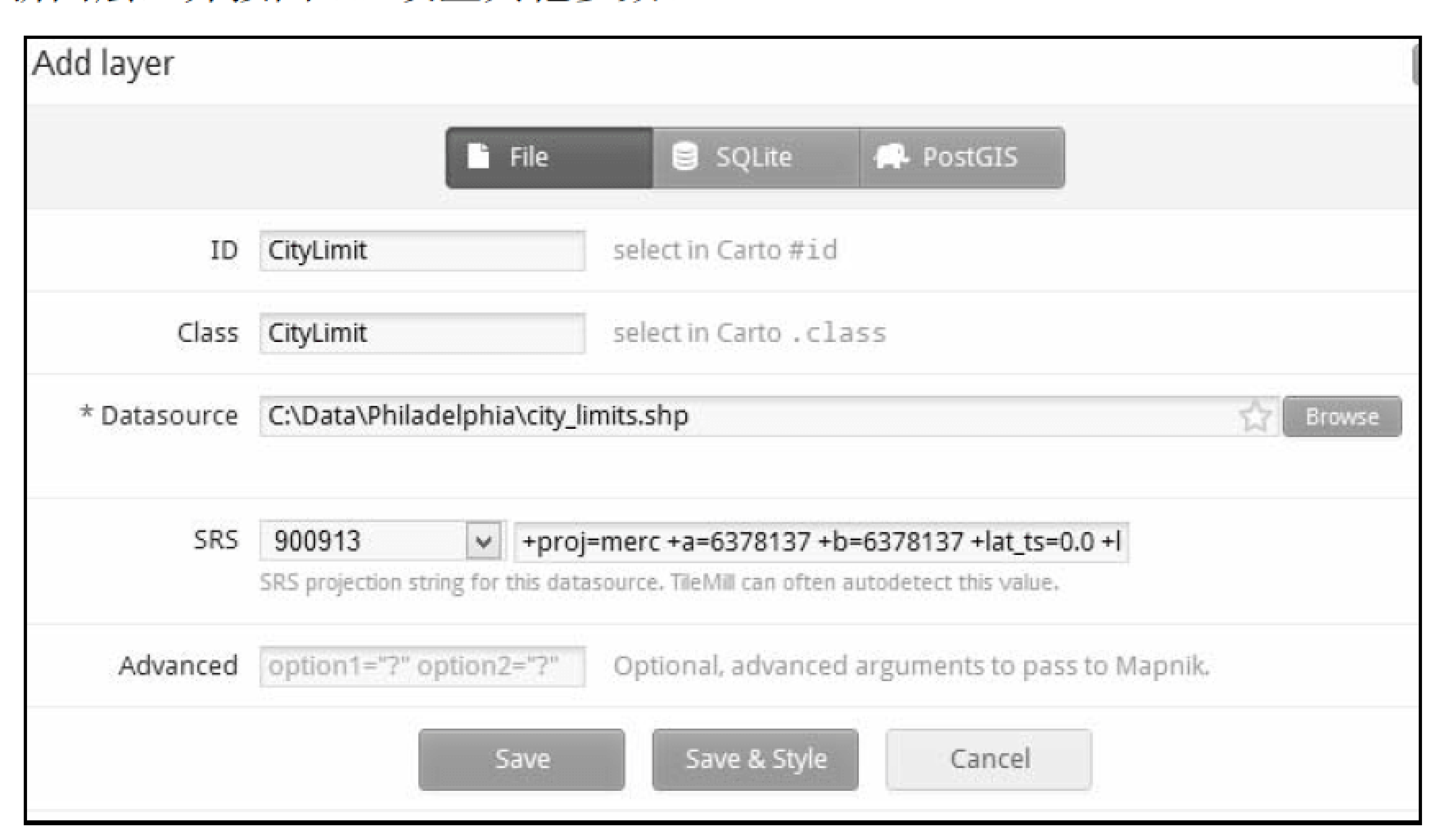


图 5.9　在 Add Layer 窗口中设置图层基本信息

TileMill 通过 Class 与 ID 的概念，允许用户设置样式规则的不同颗粒度级别。例如，如果想设置所有道路的通用规则，那么可为所有道路设置一类别，即 Class。如果想为某一子类型的道路（例如主干道）设置更为具体的规则，则可为该子类型使用一个 ID。由于这里只需要为城市边界图层设置一个样式，因此将 ID 与 Class 设置为同样的名称来。事实上，为了方便，本实践中每个图层的 ID 与 Class 都使用相同的名称。

此外，当使用 EPSG:3857 投影的数据时，必须将空间参考设置为 900913 投影代码。前面已经介绍了这两个代码的投影是一致的。

完成设置新图层的基本信息后，单击“Save & Style”按钮。

（5）放大到图层范围。

TileMill 中默认起始视图使用的是第二级，对于费城来说，比例尺太小。需要在图层管理器中单击“Zoom to extent”按钮，将地图放大到城市边界图层的范围。

（6）设置图层样式。

在 CartoCSS 代码窗口中使用如下代码，设置城市边界图层的样式。

```
#CityLimit {
    line-color:#88789e;
    line-width:3;
}
```

编辑完成后进行保存，此时 TileMill 界面如图 5.10 所示。

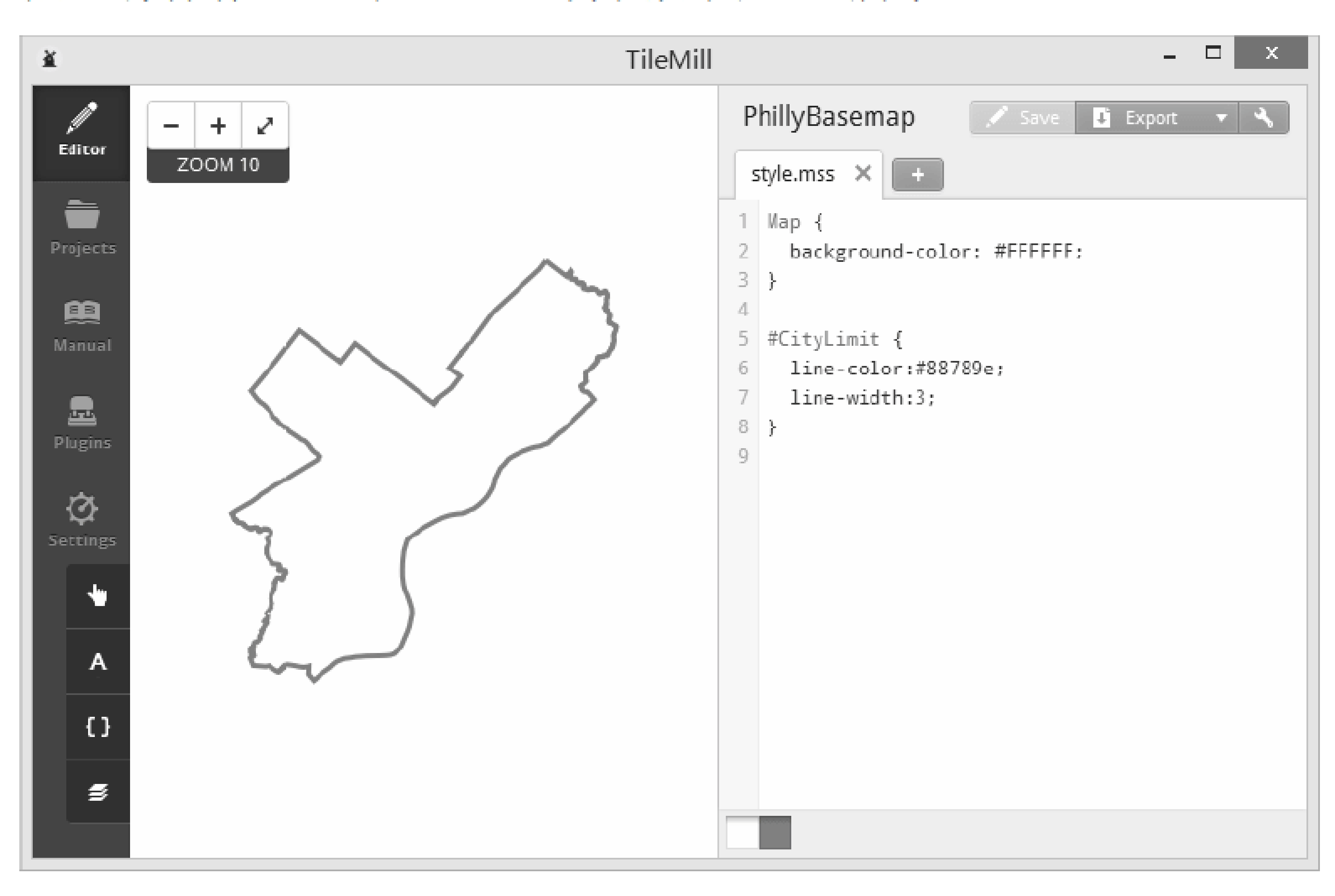


图 5.10　设置了城市边界图层样式后的地图显示

（7）设置 waterways.shp 图层的样式。

在地图中增加 waterways.shp 图层，将其样式设置为如下代码。

```
#Waterways {
    line-width:1;
    line-color:#89aceb;
}
```

（8）设置 natural.shp 图层样式。

在地图中增加 natural.shp 图层，将其样式设置为如下代码。

```
#Natural {
    [type='park']{
        polygon-opacity:1;
        polygon-fill:#ae8;
    }
    [type='riverbank']{
        polygon-opacity:1;
        polygon-fill:#89aceb;
    }
    [type='water']{
```

```
            polygon-opacity:1;
            polygon-fill:#89aceb;
        }
}
```

从上面的代码可以看出，如果只想显示部分要素，则可增加过滤器。在 natural.shp 中还包含许多其他类型的的要素，上述代码指定只显示了公园、河堤与水体要素。

（9）设置 Neighborhoods.shp 图层样式。

在地图中增加 Neighborhoods.shp，并将其样式设置为如下代码。

```
#Neighborhoods[zoom>12] {
    text-name:[NAME];
    text-face-name:"Arial Black";
    text-fill:#88789e;
    text-size: 12;
    text-character-spacing: 2;
    text-transform: uppercase;
}
```

在上述代码中，我们只指定了文字的属性。因此社区的界线将被隐藏，只显示社区名称。

当在 TileMill 中标注一图层时，需要指定包含标注文本所在的字段。对于本社区图层，依据的是 NAME 字段来标注，因此设置了“text-name:[NAME]”。

此外，上述代码还使用了缩放级别过滤器，将样式的作用范围限制在特定的地图缩放级别上。在上面的例子中，样式块中定义的样式只在地图缩放到 12 级以上时发挥作用。

在地图预览窗口中缩放地图，确保地图缩放到 12 级以上时标注才显示。如图 5.11 所示。





图 5.11　使用缩放级别过滤器

（10）设置主要道路的样式。

在地图中增加 roads.shp 图层，将其 ID 与 Class 都设置为 MajorRoads，样式代码如下。

```
#MajorRoads{
    [type='motorway']{
        line-width:3;
        line-color:#606060;
    }
    [type='trunk']{
        line-width:3;
        line-color:#606060;
    }
    [type='primary'] {
        line-width:2;
        line-color:#838383;
    }
}
```

上面的设置仅符号化了主要道路。在 OpenStreetMap 中包含了许多不同类型的道路，可以使用不同的方式进行样式化。我们这里使用的方式是，为了显示次要道路，重新加入道路图层，并将其放置在主要道路之下。这样虽然引入了一些冗余，但是代码显得很简单。

（11）设置其他道路的样式。

在地图中重新增加 roads.shp 图层，这次将其 ID 与 Class 都设置为 Roads，样式代码如下。

```
#Roads[zoom>12]{
    line-width:1;
    line-color:#b6b6b6;
}

#Roads[zoom>14]{
    line-width:1;
    line-color:#b6b6b6;
    text-name:[name];
    text-face-name:"Arial Regular";
    text-fill:#838383;
    text-size: 11;
    text-placement: line;
    text-min-path-length:100;
    text-avoid-edges:true;
    text-min-distance:50;
    text-dy: 6;
    text-max-char-angle-delta: 15;
```

```
}
```

当地图放大到 14 级以上时，将显示路名标注。

（12）设置铁路图层的样式。

在地图中增加 railways.shp，其样式代码如下。

```
#Railroads{
    line-width:1;
    line-color:#d2bcb0;
}

#Railroads[zoom>15] {
    ::line, ::hatch { line-color: #d2bcb0; }
    ::line { line-width:1; }
    ::hatch {
        line-width: 4;
        line-dasharray: 1, 24;
    }
}
```

（13）调整图层顺序。

在图层管理器中，将鼠标移动到某图层行的最前面的图标上，按住鼠标左键，便可上下移动图层顺序。将地图中按图 5.12 调整图层顺序。

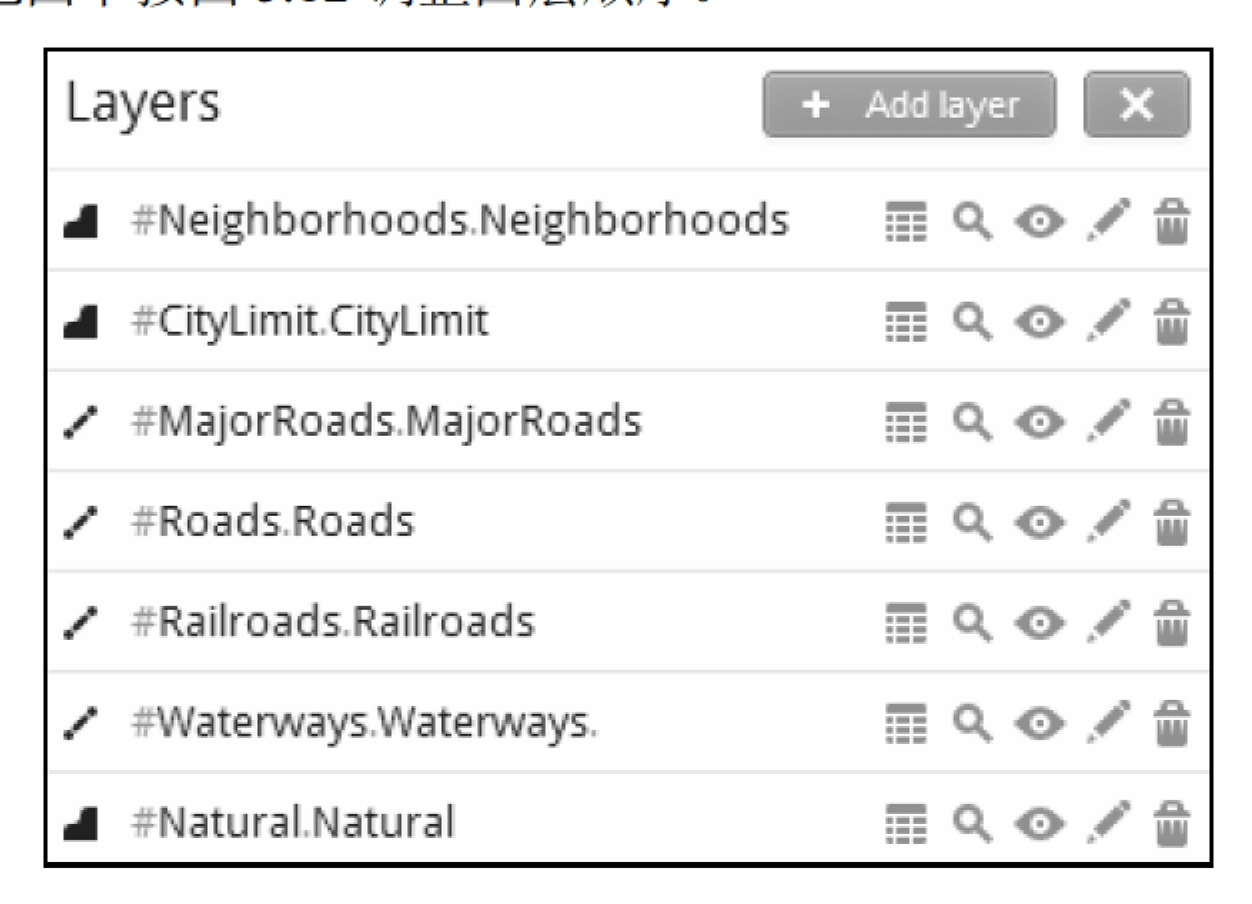


图 5.12　地图中图层的顺序

至此，我们就完成了地图设计。下一步是生成地图切片。

5.5.2　输出与提取地图切片

一旦完成了地图的设计，而且地图在各个比例尺下的显示都满足要求，便可生成地图切片了。

（1）设置元切片的缓冲区大小。

为了避免重复标注以及很长的标注被截断，可以在 CartCSS 代码中的地图的样式增加一个设置元切片的缓冲区大小，修改后的内容如下。

```
Map {
    background-color: #FFFFFF;
    buffer-size: 512;
}
```

（2）选择输出地图切片的格式。

在 TileMill 中单击“Export”按钮，选择其中的 MBTiles。MBTiles 是 MapBox 的一种将整个切片存储为一个 SQLite 数据库的格式。我们最后需要将其解压为一个一个独立的 PGN 图片。

（3）设置地图切片的范围与中心点。

在左边的地图视图中，将地图放大到费城基本填充地图窗口。然后按住 Shift 键与鼠标左键并拖动鼠标，绘制一个覆盖费城城市边界的矩形。确保地图放大到一个合理的大比例尺下，避免创建很多周边空白的图像。

然后使用鼠标右键在地图中部单击，设置中心点。

这时地图视图如图 5.13 所示。

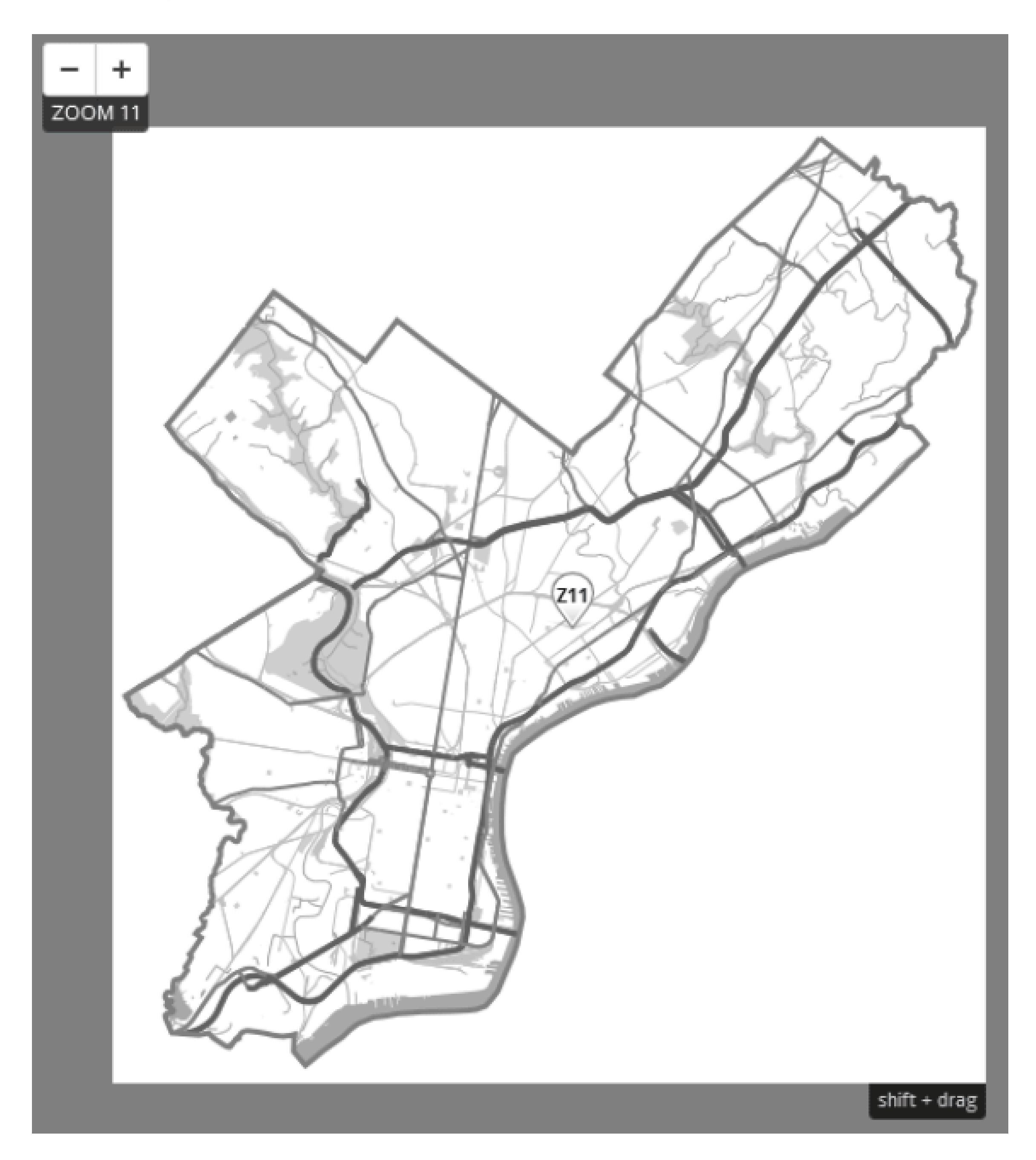


图 5.13　设置地图切片的范围与中心点

（4）设置地图切片级别。

在右边的窗口中，将缩放滑动条设置为 0~17，如图 5.14 所示。在移动滑动条时，注意观

察切片数量的变换，特别是在大比例尺级别下时。

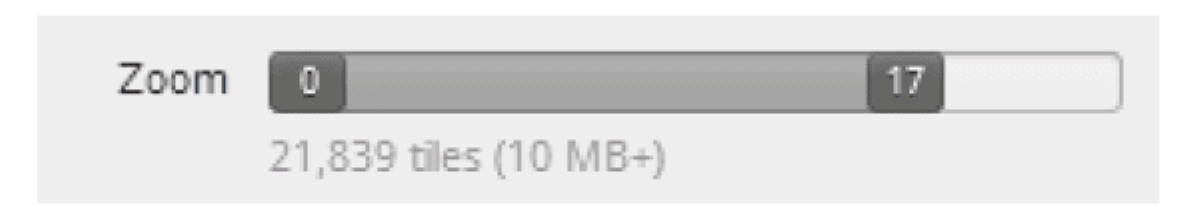


图 5.14　通过缩放滑动条设置切片级别

对于其他参数不需要修改。

（5）输出地图切片。

在右边的窗口中单击“Export”按钮，显示“View exports”窗口。在该窗口中，可观察到切片生成进度。

为了能从.mbtiles 文件中提取地图切片，需要使用名为 MBUtil 的工具。

（6）安装 MBUtil。

从“https://github.com/mapbox/mbutil”下载 ZIP 格式的文件，或直接使用下载文件的 Tools 文件夹中 mbutil-master.zip 文件。

将其解压到一个简单路径中，例如“C:\mbutil”。

（7）确保已安装 Python。

由于 MBUtil 需要 Python，因此需要确定在你的计算机上是否已经安装了 python.exe。可以查看是否有类似“C:\Python34”的文件夹。如果没有找到 python.exe，则可以从 python.org 网站下载并安装 Python。

在本实践中，假设 Python 的路径为“C:\Python27\python.exe”。如果读者使用了其他路径来安装 Python，在后面的使用时，请调整为自己的 python.exe 所在路径。

（8）复制地图切片。

在 Windows 的资源管理器中，进入到“文档\MapBox\export”文件夹，找到 PhillyBasemap.mbtiles 文件，这就是地图切片。将其复制到一个简单的路径中，例如 C:\Data\Philadelphia。

（9）提取地图切片。

打开一个命令提示符，输入并执行如下命令：

```
c:\python34\python.exe c:\mbutil\mb-util c:\data\Philadelphia\PhillyBasemap.mbtiles c:\data\Philadelphia\PhillyBasemap
```

在上面的命令行中，第一个参数是 Python 的路径，第二个参数是 MBUtil 工具的路径，第三个参数是压缩的地图切片的路径，最后一个参数是解压后切片存储的路径。读者可根据实际情况调整路径。解压缩完成后，便可在“C:\Data\Philadelphia\PhillyBasemap”路径中看到一系列的非压缩切片，如图 5.15 所示。

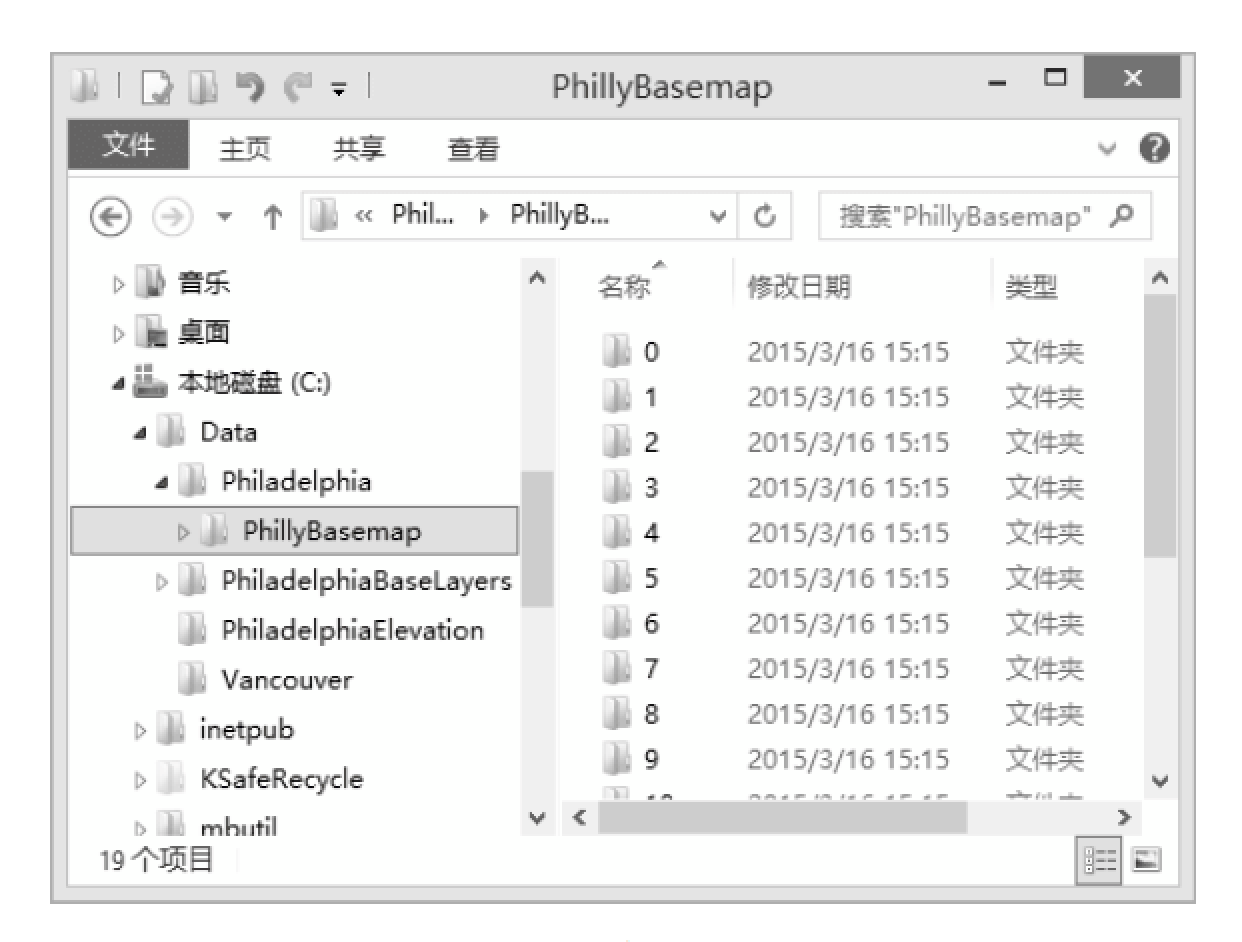


图 5.15　解压缩后的地图切片

5.5.3　发布与测试切片

仔细查看解压缩后的地图切片，可以看到切片图像严格按照“级别\列\行”的方式来组织。当前绝大多数的地图 API 都能使用该结构，因此需要做的就是将这些切片放置到 Web 服务器中。对于 Windows 系统，在计算机中安装 IIS 非常简单，我们这里就以 IIS 为例来发布地图切片。

（1）将地图切片发布到 IIS 中。

将“C:\Data\Philadelphia\PhillyBasemap”文件夹整体复制到 IIS 使用的文件夹中，即“C:\inetpub\wwwroot”文件夹。

在网页浏览器中输入类似“http://localhost/PhillyBasemap/15/9555/12400.png”的地址，查看是否能正确显示图片，如图 5.16 所示。

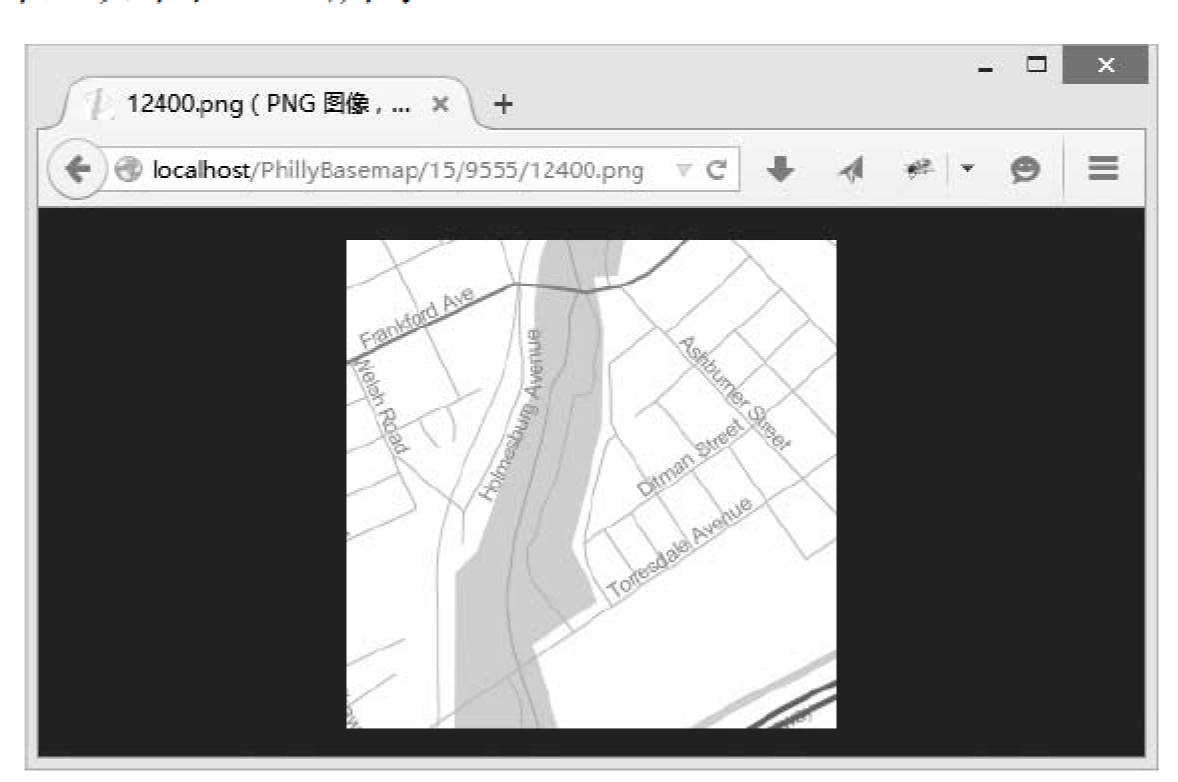


图 5.16　直接通过地址测试地图切片

（2）使用 ArcGIS.com 测试地图切片。

下面可以将该地图切片服务加入到某一在线地图中，以便在地图环境中测试。这里介绍 ArcGIS.com 地图查看器，它是基于浏览器的 Web 地图制作软件。当用于试验或与公众分享地

图时是免费使用的。

在浏览器地址栏中输入“http://www.arcgis.com/home”，然后单击“创建地图”连接，进入“我的地图”页面。在窗口的地图部分已经加入了 ESRI 提供的地图。

在工具栏中单击“添加”按钮，选择其中的“从 Web 添加图层”。如果工具栏中没有“添加”按钮，则需要先选择页面中右上角的“修改地图”。

在“从 Web 添加图层”对话框中，首先在数据类型中选择“切片图层”，将 URL 设置为“http://localhost/PhillyBasemap/{level}/{col}/{row}.png”。接着设置标题与制作者名单。然后单击“设置切片范围”按钮，在新窗口中绘制一个包含费城的矩形，并不需要精确的坐标。新图层的参数如图 5.17 所示。



图 5.17　设置新图层的参数

最后在“从 Web 添加图层”对话框中单击“添加图层”按钮，返回到地图窗口中。这时地图窗口中加入的已经是我们创建的费城地图了，如图 5.18 所示。在地图窗口中进行放大、缩小与漫游操作，进行测试。应当可以看到地图响应速度非常快。



图 5.18　利用 ArcGIS.com 测试地图切片

5.6 习 题

（1）阅读以下链接中的内容，深入理解元切片及其在 CartoCSS 中的使用：

https://www.mapbox.com/tilemill/docs/guides/metatiles/

（2）在 TileMill 中，单击左边工具条的帮助按钮，打开帮助，如图 5.19 所示。通过阅读帮助，深入了解 CartoCSS 的应用。

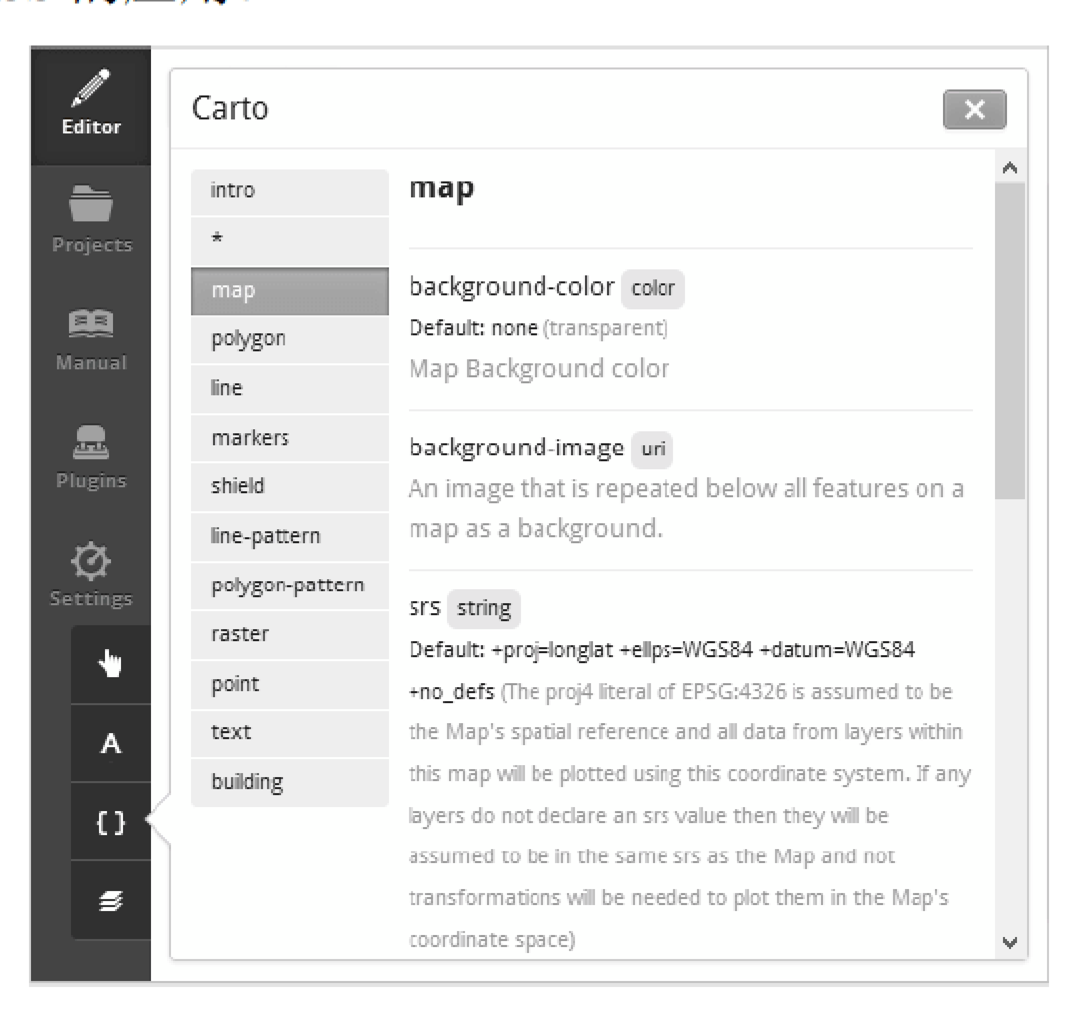


图 5.19　TileMill 中的帮助

（3）寻找 5 个使用切片地图作为背景的在线地图应用。对于每个地图，描述地图切片来源及其组织方式。

（4）利用在第 3 章习题指定收集的数据，使用 TileMill 与 CartoCSS 工具，将其发布为切片地图服务。

第 6 章

使用 Web 地图 API 访问地图服务

从本章可以学习到：

- Web 地图 API
- 使用 Web 地图 API 的基本步骤
- 查看 OpenLayers 实例
- 使用 OpenLayers 实现在切片地图上叠加 WMS

通过前面内容的学习，大家已经掌握了如何创建几种不同类型的地图服务，包括动态绘制地图服务与静态切片地图服务，同时也掌握了几种预览这些服务的几种方式。大家很可能会认为，通过预览机制，这些地图服务既难以共享，用途也不大。为了打消大家的疑虑，在介绍其他类型的空间信息服务之前，有必要介绍 Web GIS 的编程实现，为用户提供一个包含各图层的地图交互页面。可使用 Web 地图 API 帮助 Web GIS 的开发，其中一个最常用的免费开源的 API 就是 OpenLayers。本章将介绍 OpenLayers 的基本使用。

对于本章的学习，需要读者具有基本的 HTML（www.w3schools.com/html/DEFAULT.asp）与 JavaScript（www.w3schools.com/js/DEFAULT.asp）知识。对于还没掌握这些知识的读者，可通过上述两个网址来学习。

6.1 Web 地图 API

API（Application Programming Interface，应用程序接口）是编写程序的一个框架，包含了一组封装了底层代码的类与功能。例如 Web 地图 API 通常都包含表示地图与图层的类，这样开发人员就无须编写显示交互地图以及在地图中绘制图层这些底层代码了。而只需要创建一个新的地图对象 map、创建一个新的图层对象 layer，然后调用类似 map.addLayer（layer）的方法。该 API 隐藏了地图绘制任务的复杂性，让开发人员将注意力集中在应用程序的制图方面，而不是花费大量的时间研究底层逻辑。

API 也有许多不同的类型。通用的 API，如 Java 和 Microsoft .NET 框架，可以用来编写各种桌面、Web 和移动平台的程序。还有针对某些产品和功能，构建更加专业化的 API。例如，谷歌的 App Engine、Amazon Web 服务和微软的 Windows Azure，这些都是为专有的云计算环境而设计的。

在 Web 地图方面，主要的 API 有 OpenLayers、Leaflet、谷歌 Map API 以及 ArcGIS API for JavaScript。最后两个 API 是围绕特定专有平台设计的。

要注意的是，API 并不是一种编程语言，而只是使用某语言可调用的一组类与功能。一些 API 能够被多种语言调用，而有些只能被某一特定的语言所调用。

6.1.1 Web 地图 API 的选择

当准备开发一个 Web GIS 应用时，其中最重要的一个选择是使用哪个 API。如果应用程序的用户很多，该决定将可能影响随后多年的专业方面的活动与发展轨迹。

由于 API 一般都针对的是某一种语言，因此 API 的选择与开发语言和平台的选择紧密相关。开发语言与平台决定了可使用的 API 的范围。例如，如果应用程序需要运行在 Andriod 平台的平板电脑上，那么首先要决定的是要建立一个完整的原生应用程序，还是一个简单地运行在 Android 平板 Web 浏览器的程序。要开发原生应用程序，则需要使用 Java 语言；而开发基于浏览器的程序，则有更大的灵活性，可使用 JavaScript 与 HTML 来完成，或许采用有

友好移动功能的 API（换句话说，它支持触摸手势、根据设备宽度自动调整大小等）。

从上面这个例子也可以看出，开发人员熟悉的开发语言与平台，也是选择 API 时需要重点考虑的。如果有开发人员熟悉 Java 或 Objective C，那么选择开发原生应用程序的可能性就更大。然而，有了 HTML 和 JavaScript 知识通常就足以开发基于浏览器的应用程序。大多数 FOSS 的 Web 地图 API 主要面向的是 HTML 和 JavaScript 方式，因此该方式也是本书的重点。商业供应商可能会提供更多的语言和平台的选择，如 ESRI 公司同时提供了面向 iOS、Android、Flex、Silverlight 和 JavaScript 的 ArcGIS API。然而，，近年来由于 HTML5 迎头赶上以及对插件的排斥，基于插件的 API（例如 Flex 和 Silverlight）已经逐渐被抛弃。

6.1.2　主要 FOSS 类型的 Web 地图 API

下面列举的 FOSS 的 Web 地图 API 都是利用 HTML 与 JavaScript 来构建基于浏览器的应用程序。

6.1.2.1　OpenLayers

OpenLayers（www.openlayers.org）是一个用于构建 Web GIS 应用程序的、成熟的、功能丰富的 JavaScript API。它有详细的说明文档和大量的实例代码，不过有些材料对于初学者很难理解。OpenLayers 的另一个突出特点是有一个大型的开发者社区。该社区在类似 GIS Stack Exchange 等论坛上提供了大量的提示和例子。在使用 OpenLayers 开发中遇到的所有问题，基本都会在说明文档或这些网上讨论中找到答案。

OpenLayers 支持的地图来源包括谷歌地图、Yahoo!地图和微软必应地图等，用户还可以用简单的图片地图作为背景图，与其他的图层在 OpenLayers 中进行叠加。除此之外，OpenLayers 实现访问地理空间数据的方法都符合行业标准。OpenLayers 支持 OGC 制定的 WMS 和 WFS 等网络服务规范，可以通过远程服务的方式，将以 OGC 服务形式发布的地图数据加载到基于浏览器的 OpenLayers 客户端中进行显示。OpenLayers 采用面向对象方式开发，并使用来自 Prototype.js 和 Rico 中的一些组件。

在操作方面，OpenLayers 除了可以在浏览器中帮助开发者实现地图浏览的基本效果，比如放大、缩小和平移等常用操作之外，还可以进行选取面、选取线、要素选择和图层叠加等不同的操作，甚至可以对已有的 OpenLayers 操作和数据支持类型进行扩充，为其赋予更多的功能。同时，在 OpenLayers 提供的类库当中，它还使用了类库 Prototype.js 和 Rico 中的部分组件，为地图浏览操作客户端增加 AJAX 效果。

OpenLayers 中文官方网站（www.openlayers.cn/portal.php）于 2012 年 8 月成立，是由一群 OpenLayers 爱好者共同维护的，内容包括 OpenLayers 中文 API 和中文帮助文档，以及 OpenLayers 源码分析、OpenLayers 扩展开发、OpenLayers 相关工具、OpenLayers 3D 和 Openlayers Mobile 等。

6.1.2.2 Leaflet

Leaflet（www.leafletjs.com）是一个新兴的 FOSS 地图 API，目标是小巧轻便、易于上手，在移动设备上有良好的体验。由乌克兰的 Vladimir Agafonkin 带领一个专业贡献者团队开发。

在 Leaflet 中强调使用切片地图以及客户端的矢量图绘制，例如可绘制 GeoJSON。对于只需要使用切片地图以及客户端矢量绘图功能的简单 Web GIS 应用，Leaflet 绝对是一个最佳的选择。

Leaflet 设计坚持简便、高性能和可用性好的思想，在所有主要桌面和移动平台能高效运作，在当前浏览器上会利用 HTML5 和 CSS3 的优势，同时也支持旧的浏览器访问。支持插件扩展，有一个友好、易于使用的 API 文档和一个简单的、可读的源代码。但是实例却比较少，例如在 API 文档中有支持 OGC WMS 图层类型的描述，但是却没有提供相应的实例或代码。

6.1.2.3 D3

D3（www.d3js.org）是一个 FOSS 数据可视化库，常用于绘制图表，但是同时也包含了许多地图功能。D3 的一个突出特点是将数据绑定到页面的文档对象模型（Document Object Model，DOM）上，然后应用数据驱动转换到文档，从而实现数据灵活地动画与变换效果。例如可以用 D3 从数组生成 HTML 表格，或者使用相同数据平滑和动态创建一个 SVG 图表。

虽然对于初学者来说，D3 的学习曲线非常陡峭，但是对于要将地图与图表相结合的 Web GIS 应用来说也是一个比较好的选择。而且 D3 支持非 Web 墨卡托投影，下面是其中的一个可运行的实例：

http://mbostock.github.io/d3/talk/20111018/azimuthal.html

6.1.2.4 Polymaps

Polymaps（www.polymaps.org）也是一个主要面向数据可视化用户的简单 FOSS 地图库，在地图风格化方面有独到之处，类似CSS样式表的选择器。主要用途是将地图切片与GeoJSON等来源的矢量要素进行混搭。不过， Polymaps 也可将栅格图像进行仿射变换，并叠加在切片地图中（www.polymaps.org/ex/transform.html）。此外，K 均值聚类实例（polymaps.org/ex/cluster.html） 也展示了 Polymaps API 的另一个独特的特点——动态生成大量的点。

6.1.3 主要的商业 Web 地图 API

当前，商业软件公司创造了几个专用的 Web 地图 API 也已经变得非常流行。在这里，“专用”表示该 API 源代码不能下载，或不允许修改，或未经付费不能部署。

6.1.3.1 谷歌地图与必应地图 API

通过谷歌地图 API（www.developers.google.com/maps），可以将开发者自己的数据叠加

在来自谷歌切片地图上。这些叠加的数据通常使用 KML 文件方式提供，并且在客户端以可交互的矢量图形方式绘制。开发者可以使用自定义的符号对这些图形重新样式化，还可以实现用户点击某图形时，弹出一个小窗口或图表显示额外的属性信息。

由于许多用户所使用第一个互联网地图应用通常是谷歌地图，习惯了谷歌地图的导航方式与地图风格，因此使用谷歌地图 API 创建 Web GIS 的最大优点，是该应用看起来有谷歌地图的感觉，即使这是嵌入在一个陌生的第三方应用程序。虽然谷歌地图 API 与上述的 FOSS 地图 API 相比，并没有更强大或更容易使用，然而它提供了非常完整的 API 说明文档，并有一个拥有众多开发人员的开发者社区。

使用免费的谷歌地图 API 的应用程序必须是可公开访问的，并且每天不会产生超过 25000 次地图加载。不符合该条件和其他标准（https://developers.google.com/maps/licensing）的 Web GIS 应用程序必须购买商业授权的谷歌地图 API。

在世界范围内，另一家大型商业地图提供商是微软的必应地图（www.microsoft.com/maps/），其提供的用于 Web 和移动应用程序的 API，与谷歌地图 API 非常相似。与谷歌一样，必应地图同时提供了免费使用方案和必须在各种使用情况下（www.microsoft.com/maps/Licensing/licensing.aspx）购买的企业许可证。它们之间的一个区别是，必应地图 API 并不太重视 KML 使用，这是由于 KML 格式是由谷歌推广的，谷歌是用于创建 KML 文件的主要平台。

谷歌与必应地图 API 在选址、路径规划等应用程序中得到广泛的应用，典型的例子有房地产应用（trulia.com）、商业选址（yelp.com）和教堂选址（lds.org/maps）等，但是许多互联网 Web GIS 应用也正开始采用 FOSS 作为替代方案。例如 Craigslist（craigslist.org）就采用了 Leaflet 结合 OpenStreetMap 的方式展示房地产搜索结果。

6.1.3.2　ArcGIS APIs

ESRI 为 ArcGIS 平台创建了一系列的 Web 地图 API，其中有一些比谷歌地图 API 以及许多 FOSS 地图 API 提供了更丰富的功能。这些 API 支持多种语言与平台，例如 JavaScript、Flex、Silverlight、iOS 与 Android 等。理论上，虽然这些 API 功能应该基本一致，但是实际上在成熟度与应用的广泛性存在较大的差异。ArcGIS API for JavaScript 是其中功能最全、使用最广泛的 API。

ArcGIS API 主要用于访问 ArcGIS 在线（www.esri.com/software/arcgis/arcgisonline）、ArcGIS 门户网站（www.esri.com/software/arcgis/arcgisserver/extensions/portal-for-arcgis）以及 ArcGIS Server 发布的 Web 服务。但是，一些 API 也可以访问 OGC 服务、KML 以及通用的切片地图（例如在第 5 章中介绍的用 TileMill 创建的地图）。

这些 API 对于开发与教学用途是免费的，但是如果销售基于这些 API 创建的应用程序或者在其中嵌入广告，那么则需要付费。

6.2 使用 Web 地图 API 的基本步骤

如前所述，项目情况和需求决定着 API 的选择。对于不同的项目可能需要使用不同的 API，而且各 API 都各有优缺点，在一个项目中常常需要混用多个 API。因此作为一个程序员，需要同时掌握多个 API 的使用。那么重要的是要了解这些 API 与语言背后的通用结构、模式及架构，这样便可在使用过程中学习新的 API。技术总是在不断变化，如果将自己绑到某一单一的开发模式，就会限制编程工具的选择。

下面将介绍使用上述 Web 地图 API 的一些通用模式与过程。在深入介绍 OpenLayers 之前，将这部分内容独立出来介绍，目的就是让读者知道在编写代码时哪些内容并不是 OpenLayers 所独有的。

6.2.1 引用 JavaScript 与样式文件

在使用 Web 地图 API 编写功能之前，需要在 HTML 页面中增加一个<script>标签，指向 Web 地图 API 的 JavaScript 文件。在引用 JavaScript 时要清晰地意识到：所引用的 JavaScript 文件越多，那么页面加载所花费的时间越长。一些 API 比较小（因此有像 ModestMaps 名称），但是提供的功能也可能要少。请注意，OpenLayers 是一个功能强而且加载时间也长的最大的 API。当采用较大的 API 时，一些开发人员构建和引用只包含自己需要功能的缩小版 API。

引用 API 的方式有几种。一种是将 API 下载并部署到自己的服务器上，这样做的好处是将加载时间减到最小，而且可以自定义 API。另一种方式是引用他人服务器上的 API。内容分发网络（Content Delivery Network，简称 CDN）站点专门用于存储通用 API。对于在 CDN 中 OpenLayers 的引用方式如下：

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js"> </script>
```

不过，一般 API 开发者也提供 API 的直接的引用。例如对于 OpenLayers，也可使用如下方式来引用：

```
<script src="http://openlayers.org/api/OpenLayers.js"> </script>
```

在本书中为了简便，直接通过 CDN 的方式来引用 OpenLayers。但是如果是开发内部局域网应用程序，或者需要自定义 API，那么则需要下载 API 的源代码并部署到自己的服务器中。

许多 Web 地图 API 在提供 JavaScript 的同时也提供了一些样式表。通过 CSS 文件引用的方式使用这些样式表。例如从 CDN 上引用 OpenLayers 样式表的代码如下：

```
<link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/theme/default/style.css" >
```

6.2.2　地图 div 与对象

当需要在页面中放入一个地图时，通常需要在页面中增加一个 HTML 的<div>元素用于显示地图，然后使用 API 创建一个地图对象，并将地图对象与增加的 div 元素建立连接。

在页面的<body>部分增加一个<div>元素的代码如下：

```
<body>
    <div id="map"></div>
</body>
```

然后在页面的 JavaScript 代码中创建一个 OpenLayers.Map 对象，并建立与 div 的关联。OpenLayers.Map 的构造函数将 div 的 ID 作为作为参数。代码如下：

```
var myMap;
myMap = new OpenLayers.Map("map");
```

通过上述方式，可以在页面中同时放置多个地图。每个地图有自己的 div 与 OpenLayers.Map 对象。

地图的空间参照系会影响地图的显示，并决定使用的坐标格式。下面是一个稍微复杂的例子，显示了如何显式使用 Web 墨卡托投影定义坐标系统：

```
var toProjection = new OpenLayers.Projection("EPSG:900913"); // Web 墨卡托投影
var map;
map = new OpenLayers.Map("map", {projection:toProjection});
```

在调用 OpenLayers.Map 的构造函数时，可以将一个 JavaScript 对象作为可选的参数传进来。由于该参数是一个 JavaScript 对象，因此它可以包含 JavaScript 中的所有事物，包括字符串、数字、数组、日期以及 JavaScript 对象等。在上面的例子中，先创建了一个表示 Web 墨卡托投影的对象，然后在创建地图对象时将其赋给一匿名的 JavaScript 对象的 projection 属性，从而将整个地图的空间参照系设置为 Web 墨卡托投影。

地图对象包含了增加图层、得到当前图层集合、地图窗口操作等。对于大多数的 Web 地图 API，地图是最重要而且是功能最强大的对象。当学习一个新的地图 API 时，首先需要认真查看其关于地图对象的文档，理解该对象包含的方法以及如何调用其中最常用的方法。

6.2.3　Layer 对象

大多数 Web 地图 API 一般都提供了多种方式定义图层对象，然后将图层对象逐一加入到地图对象中，用以创建混搭应用。对于图层需要特别注意的是，在某种意义上，图层就是代表 WMS 或切片地图的一个 Web 服务，其本身也可能包含许多数据图层。但是，只需要创建一个图层对象，就可将该类型的服务加入到地图中。此外，其他图层对象可以引用一个数据文件，例如 KEML 或 GeoJSON。

在许多 Web 地图 API 中，图层对象通常是一抽象的类，提供了一系列通用属性，该类不能实例化，开发人员只能实例化其更具体的子类。OpenLayers.Layer 类的帮助说明文档地址如下：

```
dev.openlayers.org/releases/OpenLayers-2.13.1/doc/apidocs/files/OpenLayers/Layer-js.html
```

图层类包含了所有图层都有的投影、单位和比例尺等属性。图层类的子类有 30 多个，常用的有 OpenLayers.Layer.WMS、OpenLayers.Layer.Grid 与 OpenLayers.Layer.Vector 等。

当在 OpenLayers 中创建一个新的图层对象时，通常需要提供该图层的 URL 或是包含数据源的文件路径。不过，新建图层对象后并不会立即显示该图层，需要调用地图对象的 addLayer 方法将其加入到地图中才能显示。下面的代码展示了如何创建一个 WMS 图层，并叠加在底图上：

```
layer = new OpenLayers.Layer.WMS(
  "WMS", "http://localhost:8080/geoserver/philadelphia/wms",
  {
    LAYERS: 'philadelphia:FarmersMarkets',transparent: true
  },
  {
    singleTile:true,
    isBaseLayer: false
  }
);
map.addLayer(layer);
```

在创建 WMS 图层对象时，最主要是提供一个 URL 与一个图层名，而其他属性像 singleTile 与 isBaseLayer 仅仅是指示在地图中如何显示该图层。

地图对象的 addLayer 方法才是真正将图层加入到地图并显示。如果再调用 addLayer 并传入其他图层作为参数，那么该新加入的图层显示在地图的最顶层。

有些类型的图层，像 OpenStreetMap、必应地图，有大家都熟知的 URL，因此对于这些图层，在创建对象时 OpenLayers 并不需要开发人员提供 URL，只需要提供一个用户友好的名称即可。例如，下面的代码演示了如何在地图中加入基本的 OpenStreetMap 切片地图：

```
var osm = new OpenLayers.Layer.OSM( "Simple OSM Map");
map.addLayer(osm);
```

6.2.4　图层样式化机制

那些在服务器端绘制的图层，例如切片地图与 WMS 图像，以及应用了样式，但是对于由浏览器绘制的图层，例如 GeoJSON 或 GeoRSS，必须定义如何样式化这些图层。Web 地图 API 中通常会提供一组属性用于指定浏览器如何绘制这些图层。这组属性包括填充颜色、填

充宽度和轮廓线宽度和轮廓线颜色等。许多 API 允许使用自定义的图像符号化点对象，而不是仅仅放置一个简单的点。

下面的 OpenLayers 代码演示了如何将一个包含食品商店的 GeoJSON 图层加入到地图中，并使用保存在一个名为 grocery.svg 的 SVG 文件中的购物车图标来样式化。该图标也可以是一个 PNG 文件或其他类型的图像。

```
var fromProjection = new OpenLayers.Projection("EPSG:4326");
var groceryLayer = new OpenLayers.Layer.Vector("Grocery", {
  projection: fromProjection,
  strategies: [new OpenLayers.Strategy.Fixed()],
  protocol: new OpenLayers.Protocol.HTTP({
    url: "supermarkets.geojson",
    format: new OpenLayers.Format.GeoJSON()
  }),
  style: {
    externalGraphic: 'svg/grocery.svg',
    graphicWidth: 25,
    graphicHeight: 25,
    graphicYOffset: 0
  }
});
map.addLayer(groceryLayer);
```

与本例相似的运行结果如图 6.1 所示。



图 6.1　使用图标样式化点图层

6.2.5　事件与交互元素

只有通过交互元素，才能使地图不再是页面中的一张静态图片。上面描述的地图与图层对象都可以通过程序来响应用户的行为，例如鼠标单击。用户的行为称为事件，响应事件所调用的代码称为事件处理程序。通常事件都会为事件处理程序提供事件相关的参数。

例如，可设置让地图监听用户鼠标单击事件，然后编写一个事件处理程序，输入该程序的参数是用户鼠标单击处的屏幕坐标。在处理程序中，首先需要将屏幕坐标转化为地图坐标，然后将该坐标写入到 HTML 页面中的某一标签中，这样用户便能看到单击处的坐标。当然，

还可以将该事件处理程序绑定到鼠标移动事件上，那么便能在标签中时刻显示用户鼠标当前所在的位置坐标。

Web 地图用户通常希望获得地图中某些具体要素的更详细信息。通常的实现方式是，处理单击事件，弹出一个小窗口，在其中显示被单击要素的更多信息，如图 6.2 所示。事实上，许多 Web API 都有专门显示与处理弹出式窗口的类与方法，方便开发人员实现类似的功能。

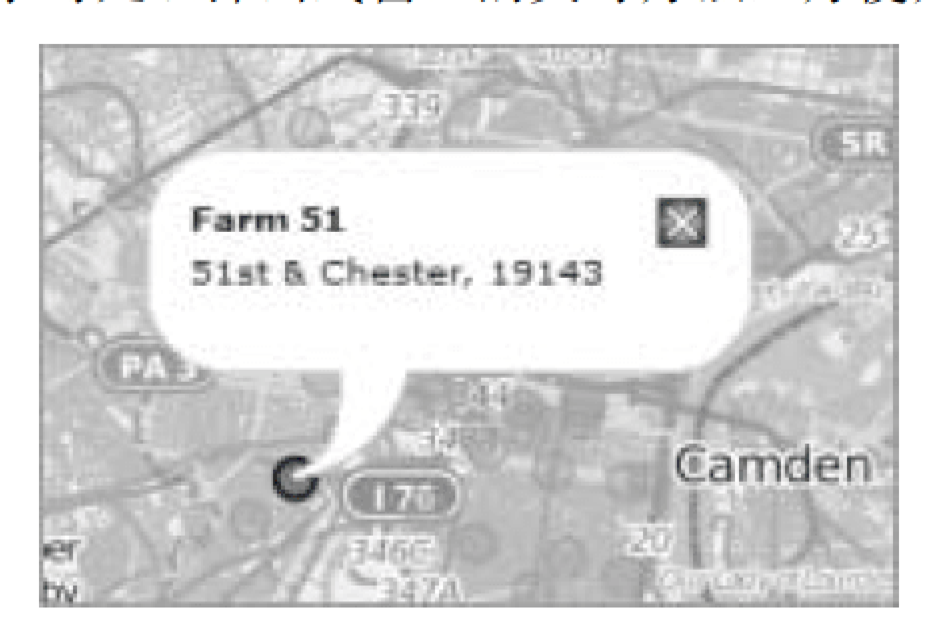


图 6.2　通过弹出式窗口显示要素更详细信息

正如图 6.2 所显示的，通常弹出式窗口只用于显示比较简单的信息。但是有时希望显示更多的内容，这对于弹出式窗口有限的空间来说过于复杂，即使 API 允许在弹出式窗口中显示很大的图像或表格，但对用户更友好的方式是在页面的另一个 div 中显示。Web 地图 API 通常都提供通过坐标点执行空间查询，查询到用户单击了那个要素，并获取该要素的属性信息。有了这些信息之后，在事件处理程序中可做进一步的应用，例如可将这些信息传递给另个一专业的 API 绘制图表来查询维基百科、查询周边销售中的房屋等。

另一个比较常用的交互是图层的显示与隐藏。要注意，在 Web 地图中，图层代表的是整个 Web 服务。由于所有的图层都绘制在切片图像中，因此对于切片地图来说，不可能切换其中某个图层的可见性。但是，可将整个切片图层关闭，也可关闭 WMS 或 GeoJSON 图层。

在 OpenLayers 中，通过地图对象可以获取其包含的所有图层对象，并且图层对象有一个 setVisibility 方法，因此要切换图层的可见性只需要很简单的几行代码。下面的示例函数用于切换指定名称的图层的可见性：

```
function toggleLayerViz(layerName){
  var layerToToggle = map.getLayersByName(layerName)[0];
  if (layerToToggle.visibility){
    layerToToggle.setVisibility(false);
  }
  else {
    layerToToggle.setVisibility(true);
  }
}
```

6.3　查看 OpenLayers 实例

学习 OpenLayers 或者其他任务 API，最好的方式就是查看并运行其开发者的实例代码，并按照自己的习惯进行一些调整与调试。在 Web GIS 应用程序的实际开发过程中，一种策略是先在互联网上寻找与应用程序想类似的例子，然后将这些例子结合在一起进行调整，并加入自己的内容，直到满足要求。

以下将通过分析几个典型的实例，加深读者对 OpenLayers 开发的理解。

6.3.1　切片地图实例

OpenLayers 实例的地址是 dev.openlayers.org/releases/OpenLayers-2.13.1/examples，这是寻找参考代码资源的最重要的地方。

可以选择几个感兴趣的页面，利用浏览器的“查看页面源代码”功能查看源代码。

访问来自 ESRI ArcGIS 在线中切片地图的实例代码地址如下：

http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/xyz-esri.html

在其页面源代码中，定位到<script>标签部分，其内容如下：

```
    var map, layer;
    function init(){
        var layerExtent = new OpenLayers.Bounds( -13758743.4295939, 5591455.28887228,
-13531302.3472101 , 5757360.4178881);
        map = new OpenLayers.Map( 'map', {'restrictedExtent': layerExtent} );
        layer = new OpenLayers.Layer.XYZ( "ESRI",

"http://server.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer/tile/${z}/${y}/${x}",
                {sphericalMercator: true} );
        map.addLayer(layer);
        map.zoomToExtent(map.restrictedExtent);
    }
```

上面的代码依次实现了如下一些功能：

（1）为地图与图层对象分别申明了变量。

（2）设置了图层最大可查看的边界坐标。这有助于防止用户在切片区域区外缩放地图。

（3）将地图 div 的 id 与边界对象作为参数，创建地图对象，

（4）创建图层对象，传递的参数有图层的名称、切片的 URL 结构，以及指示切片使用了 Web 墨卡托投影的可选属性。

（5）将图层加入到地图中。

（6）将地图放大到最大可查看边界坐标。

上述的主要代码集中放置于 init()函数中，由于在页面中有<body onload="init()">，表示页

面内容加载完成之后立即执行 init()函数。

该实例使用了 OpenLayers.Layer.XYZ 类来访问切片图层。那么请读者考虑，如何使用该类来访问我们在第 5 章中创建的费城切片地图呢？

6.3.2 WMS 实例

访问 WMS 服务最简单的实例的地址如下：

http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/lite.html

该实例演示了如何访问运行在一个公共服务器上的 WMS 服务，并将其加入到地图中。其代码如下：

```
var map, layer;
function init(){
    map = new OpenLayers.Map( 'map' );
    layer = new OpenLayers.Layer.WMS( "OpenLayers WMS",
        "http://vmap0.tiles.osgeo.org/wms/vmap0",
        {layers: 'basic'} );
    map.addLayer(layer);
    map.zoomToMaxExtent();
}
```

在整体结构上，该代码与“6.3.1 切片地图实例”中的代码完全一致，唯一的不同是图层创建方式的不一样。在本代码中，使用了 OpenLayers.Layer.WMS。要使用该类，需要在构造函数中提供图层的名称、WMS 服务的 URL，以及希望显示的图层列表。上述代码表示只需要从 WMS 服务中请求名为 basic 的图层。

6.3.3 查询实例

得到用户所单击要素的属性信息，并将其显示在一个弹出式窗口中，是一个 Web GIS 应用程序中最常用的功能。一个这样实例的地址如下：

http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/getfeatureinfo-popup.html

在该实例的 JavaScript 代码中，第一句代码如下：

```
OpenLayers.ProxyHost = "proxy.cgi?url=";
```

该行代码用于设置代理地址。当应用程序异步调用远程 Web 服务时，例如查询 WMS 服务，该请求需要通过安装在服务器上的代码，将其转发到远程 Web 服务上。

接下来是如下代码：

```
var map, info;
```

```
function load() {
    map = new OpenLayers.Map({
        div: "map",
     maxExtent: new OpenLayers.Bounds(143.834,-43.648,148.479,-39.573)
    });

    var political = new OpenLayers.Layer.WMS("State Boundaries",
        "http://demo.opengeo.org/geoserver/wms",
        {'layers': 'topp:tasmania  state  boundaries', transparent: true, format: 'image/gif'},
        {isBaseLayer: true}
    );

    var roads = new OpenLayers.Layer.WMS("Roads",
        "http://demo.opengeo.org/geoserver/wms",
        {'layers': 'topp:tasmania_roads', transparent: true, format: 'image/gif'},
        {isBaseLayer: false}
    );

    var cities = new OpenLayers.Layer.WMS("Cities",
        "http://demo.opengeo.org/geoserver/wms",
        {'layers': 'topp:tasmania  cities', transparent: true, format: 'image/gif'},
        {isBaseLayer: false}
    );

    var water = new OpenLayers.Layer.WMS("Bodies of Water",
        "http://demo.opengeo.org/geoserver/wms",
        {'layers': 'topp:tasmania_water_bodies', transparent: true, format: 'image/gif'},
        {isBaseLayer: false}
    );

    map.addLayers([political, roads, cities, water]);
    // 其他代码
}
```

上述代码初看很复杂，但细看以后发现其实逻辑也很简单，就是分别创建了 4 个 WMS 图层对象，所使用的是同一个 WMS 服务，只是对应不同的图层而已，最好将它们加入到地图中。

接下来就是查询与显示查询结果的代码，如下所示：

```
info = new OpenLayers.Control.WMSGetFeatureInfo({
     url: 'http://demo.opengeo.org/geoserver/wms',
     title: 'Identify features by clicking',
```

```
        queryVisible: true,
        eventListeners: {
            getfeatureinfo: function(event) {
                map.addPopup(new OpenLayers.Popup.FramedCloud(
                    "chicken",
                    map.getLonLatFromPixel(event.xy),
                    null,
                    event.text,
                    null,
                    true
                ));
            }
        }
    });

    map.addControl(info);
    info.activate();
```

在上述代码中，使用了 WMS 的 GetFeatureInfo 方法执行地图的查询。不过，在 OpenLayers 中使用 GetFeatureInfo 相对有点复杂。首先需要创建一个 OpenLayers.Control.WMSGetFeatureInfo 控件，并传入如下一些参数：

（1）要查询的 WMS 服务地址。

（2）该控件的标题。

（3）一个布尔变量，指示是否对隐藏图层的要素也进行查询。

（4）最后是一个事件监听器，定义了当查询执行或完成后应该执行的代码。

前 3 个参数很简单，不需要进一步解释。唯独第 4 个参数，对于初学者来说会比较困惑。

要理解第 4 个参数，首先要知道 WMSGetFeatureInfo 控件可以监听名为 getfeatureinfo 的事件。该控件处理单击或悬停事件，并使用 OpenLayers.Format 解析结果，然后激活 getfeatureinfo 事件。

从上面的代码可以看到，事件返回了一个事件对象，该对象包含了被查询要素的一些信息。事件处理函数将返回的信息放置到一个弹出式窗口中。该事件处理程序代码被封装在 function(event){ . . . }代码块中。

真正实现弹出窗口的代码是地图对象的 addPopup 方法。该方法将弹出式窗口对象作为参数。在上面的代码中，利用 OpenLayers.Popup.FramedCloud 构造函数动态创建了一个 FramedCloud 类型的弹出式窗口。创建该对象的代码如下：

```
new OpenLayers.Popup.FramedCloud(
    "chicken",
    map.getLonLatFromPixel(event.xy),
    null,
    event.text,
```

```
        null,
        true
));
```

通过上述代码可以看出，传入了以下 6 个参数：

（1）一个代表该弹出式窗口的唯一 ID。该 ID 基本不使用，因此该实例的开发者幽默地将其设置为“chicken”。

（2）弹出式窗口固定位置的经纬度。这里事件对象派上用场了，它的 xy 属性包含了该信息。

（3）窗口中内容的大小。如果设置为 null，那么弹出式窗口就会根据内容多少，自动调整大小。

（4）窗口中显示的文本内容。对于本实例，WMS 返回的是 HTML 表格元素，其中包含被查询要素的属性。事件对象的 text 属性就包含了该 HTML 内容。

（5）固定窗口的对象。

（6）一个布尔变量，指示是否在窗口的右上角显示一个红色的图标，以便关闭窗口。

6.4　实践 10：使用 OpenLayers 实现在切片地图上叠加 WMS

本实践的目标是让读者了解在 OpenLayers 的地图中叠加来自不同类型的 Web 服务。首先需要发布显示费城农贸市场的 WMS 服务，然后利用 OpenLayers 将该图层叠加在第 5 章实践中创建的切片地图上，最后增加一些相应用户单击农贸市场要素的代码，在弹出式窗口中显示更多的信息，运行效果如图 6.3 所示。

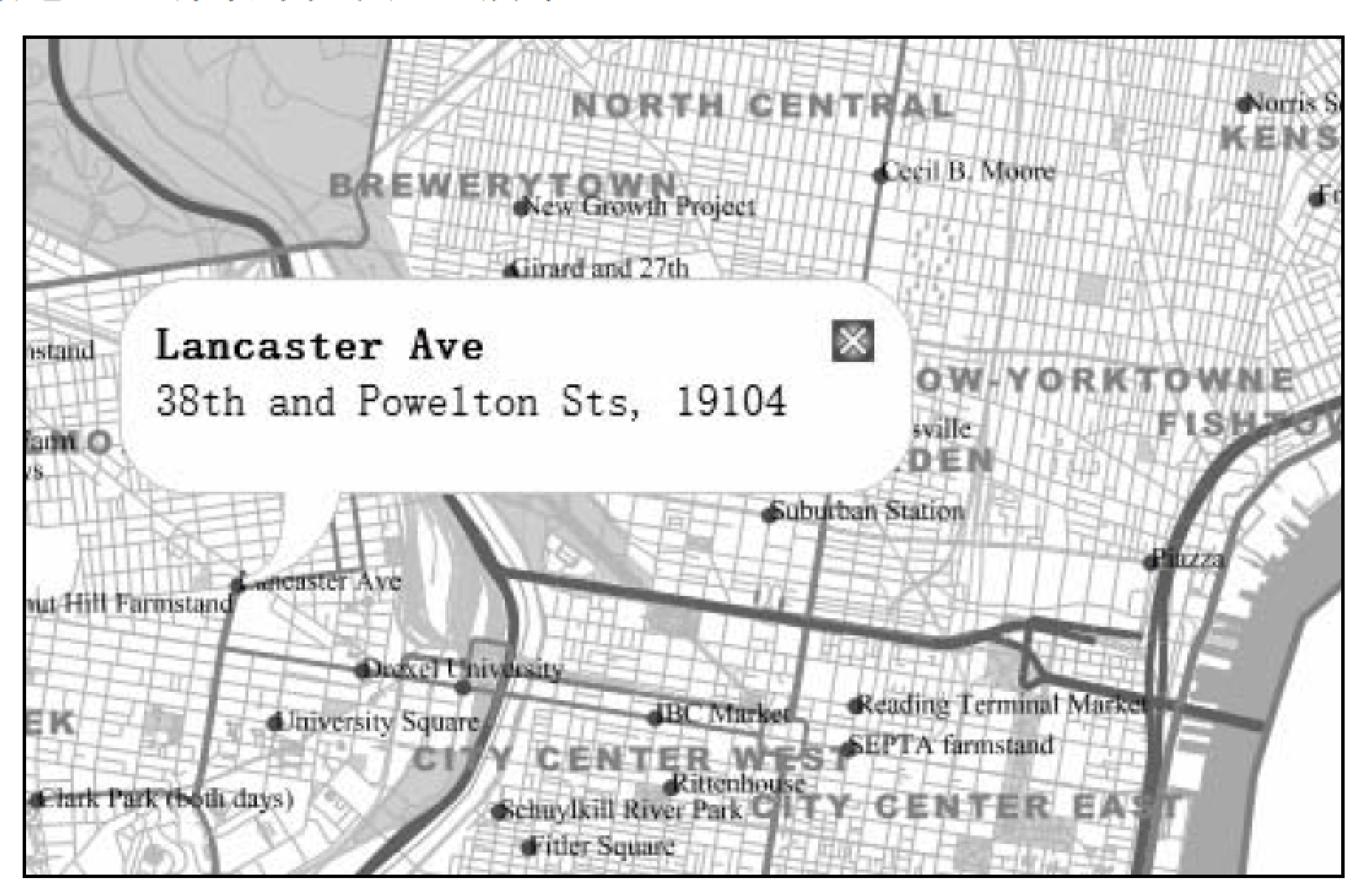


图 6.3　应用程序运行效果

6.4.1　发布专题数据 WMS 服务

第一步是将费城农贸市场点数据发布为一个美观的 WMS 服务。在本应用程序中，农贸市场 WMS 图层扮演的角色是业务图层。

（1）从下载文件的“Data\FarmersMarkets”文件夹中将数据复制到“C:\Data\Philadelphia”文件夹中。

（2）打开 GeoServer 的 Web 管理页面，按照第 4 章中介绍的方法，将上面的 FarmersMarkets.shp 农贸市场数据发布为一个图层并将其放置在 webgis 工作区中。使用 OpenLayers 预览该图层，结果显示如图 6.4 所示。

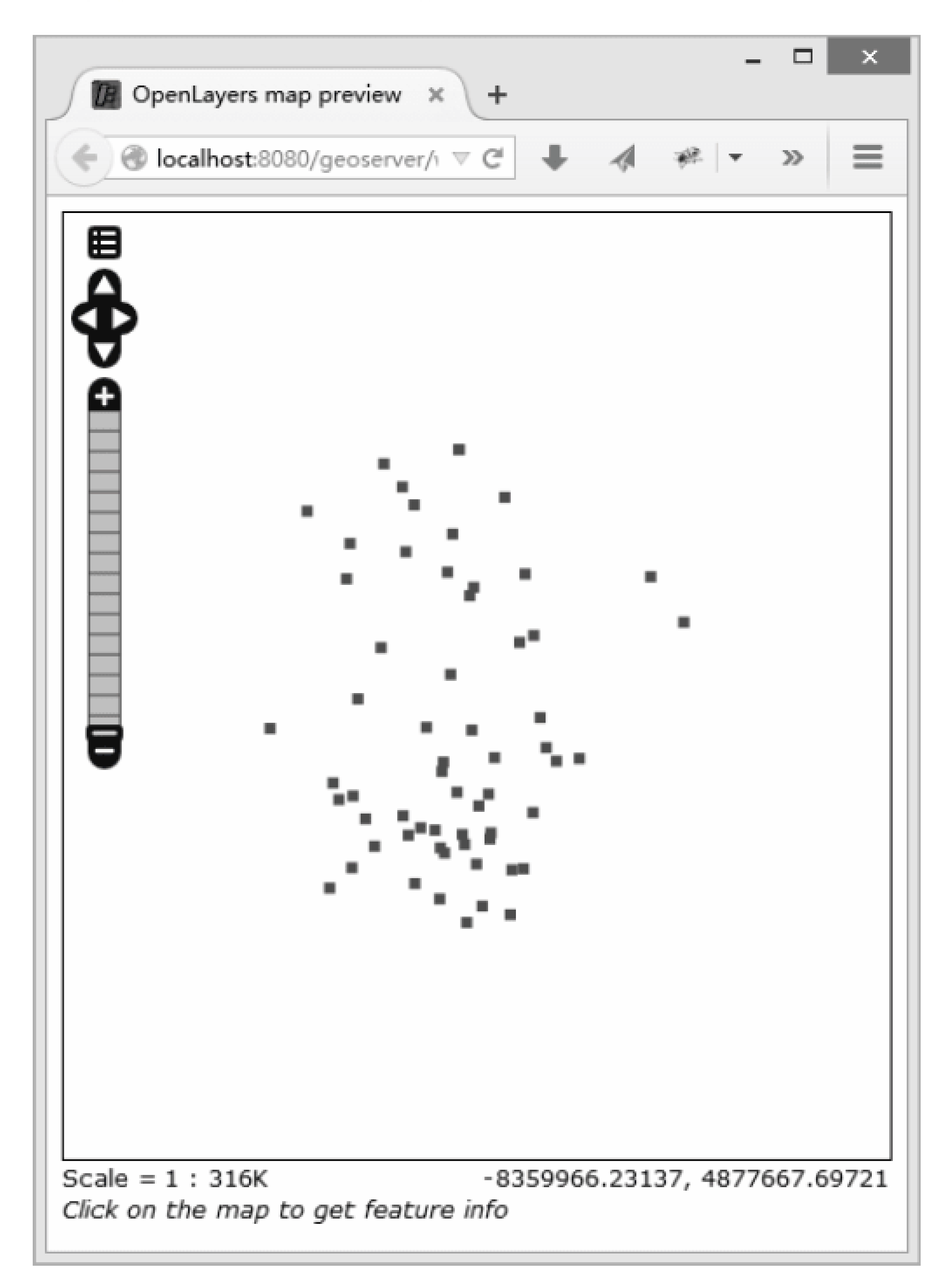


图 6.4　使用 OpenLayer 预览农贸市场图层

（3）从 SLD 说明文档中找到带“标注的点符号”例子，其 URL 地址如下：

http://docs.geoserver.org/stable/en/user/styling/sld-cookbook/points.html#point-with-styled-label

以该例子为基础，创建一个名为 point_pointwithstyledlabel 的 SLD 文件。

由于 SLD 在读包含标注名称的字段时，是要区分大小写的。而农贸市场的 Shapefile 数据中包含名称的字段是 NAME，因此需要按照如下方式修改 SLD 的代码：

```
<Label>
    <ogc:PropertyName>NAME</ogc:PropertyName>
</Label>
```

（4）按照第 4 章介绍的方法，将 point_pointwithstyledlabel 样式化图层描述符应用于农贸市场图层，将其设置为默认样式或唯一样式。

当成功设置了 SLD 以后，使用 OpenLayers 预览农贸市场 WMS 图层，显示结果如图 6.5 所示。

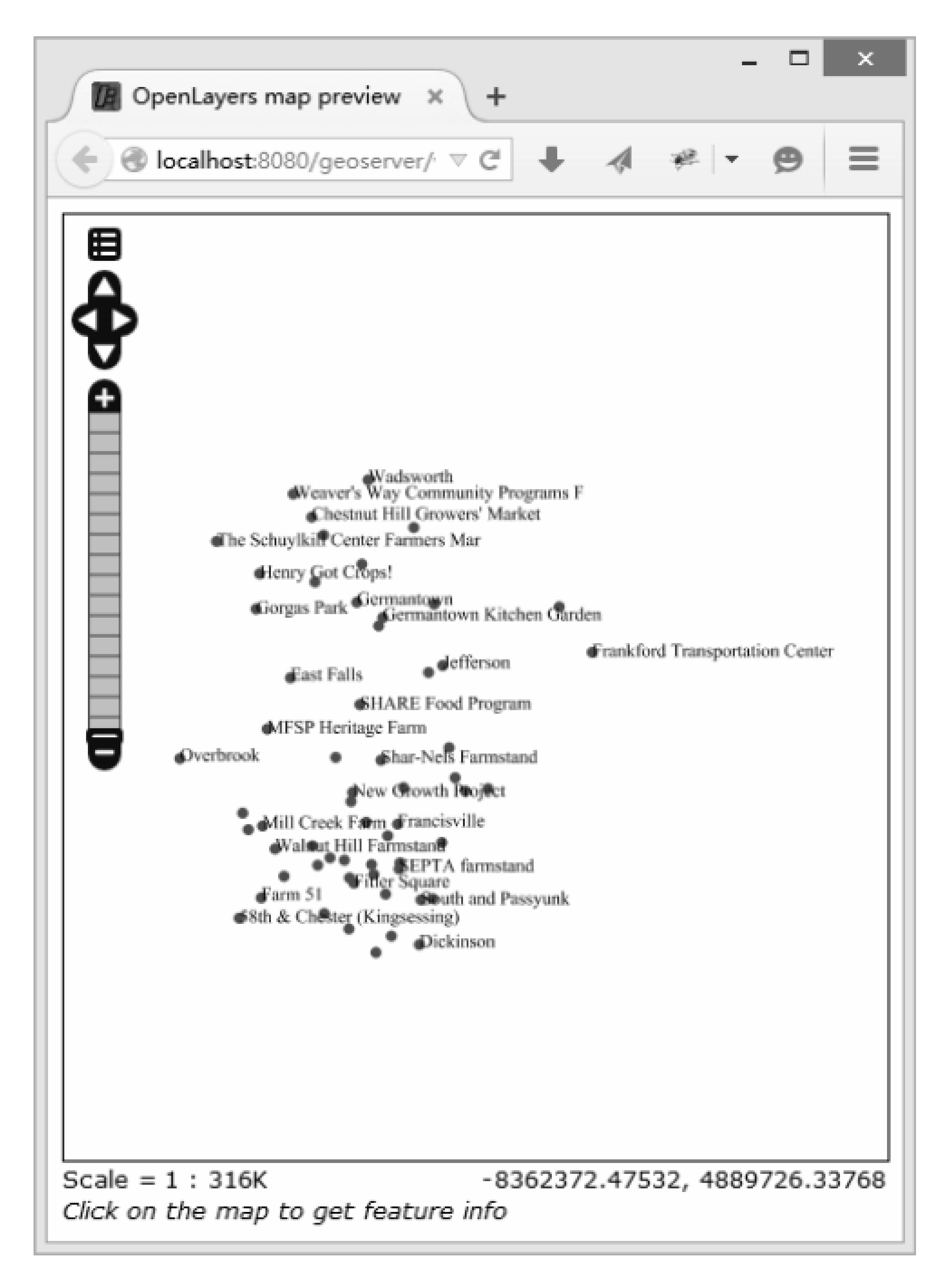


图 6.5　使用了 SLD 以后的农贸市场图层

6.4.2　准备开发环境

由于浏览器的同源策略，一般 Web 服务器不允许第三方的脚步直接调用 Web 服务，因此正如前面介绍的，需要通过服务器将客户端的请求转发到 Web 服务所在的服务器上。在 OpenLayer 中，需要设置一个代理服务器程序。如果没有设置，那么在 Firebug 或其他开发者工具则会提示图 6.6 所示的错误。

```
已阻止交叉源请求：同源策略不允许读取 http://localhost:8080/geoserver/webgis
/wms?LAYERS=webgis%3AFarmersMarkets&QUERY_LAYERS=webgis%3AFarmersMarkets&STYLES=&
SERVICE=WMS&VERSION=1.1.1&REQUEST=GetFeatureInfo&BBOX=-
8384671.013734%2C4846374.399588%2C-8345535.255258%2C4885510.158064&
FEATURE_COUNT=10&HEIGHT=512&WIDTH=512&FORMAT=image%2Fpng&
INFO_FORMAT=application%2Fjson&SRS=EPSG%3A900913&X=244&Y=134 上的远程资源。可以将资
源移动到相同的域名上或者启用 CORS 来解决这个问题。
```

图 6.6　同源策略阻止交叉源请求

不过，随着 HTML 的 CORS（Cross Origin Resource Sharing，跨域资源共享）的推出，可

以在服务器端进行设置，以便允许第三方的脚步直接访问服务器中资源。这样便可不再需要代理程序了，而且由浏览器直接访问，不通过服务器便可访问 Web 服务，因此程序响应速度肯定有很大的改进。

但是 GeoServer 默认时，并不允许跨域资源共享。需要进行一定配置，才能实现跨域资源共享。下面我们就来介绍配置过程。

（1）下载资源。

下载 http://shanbe.hezoun.com/cors.zip，或直接从下载文件的 Tools 文件夹中复制 cors.zip。

将其解压到 GeoServer 的 classes 文件夹中。如果是默认情况，那么该文件夹的路径为“C:\Program Files\GeoServer 2.6.2\webapps\geoserver\WEB-INF\classes”。

（2）配置 web.xml。

打开“C:\Program Files\GeoServer 2.6.2\webapps\geoserver\ WEB-INF\web.xml”，在其他 filter 后面，加入如下配置：

```
<filter>
  <filter-name>cross-origin</filter-name>
  <filter-class>org.mortbay.servlets.CrossOriginFilter</filter-class
  <init-param>
    <param-name>allowedOrigins</param-name>
    <param-value>*</param-value>
  </init-param>
  <init-param>
    <param-name>allowedMethods</param-name>
    <param-value>GET,POST</param-value>
  </init-param>
  <init-param>
    <param-name>allowedHeaders</param-name>
    <param-value>x-requested-with,content-type</param-value>
  </init-param>
</filter>
```

然后在其他 filter-mapping 之后，加入如下配置。

```
<filter-mapping>
  <filter-name>cross-origin</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

（3）重新启动 GeoServer。

6.4.3 页面设计与代码编写

本实践要实现的功能是在费城的切片地图上显示农贸市场的位置，通过单击某农贸市场，

可在弹出窗口中显示更多信息。

（1）新建一个目录，例如 Walkthrough10，然后在其中创建一个空的文本文件，将其保存为 Markets.html。

如果读者熟悉使用 Visual Studio 或 Eclipse 等集成开发环境，那么尽可使用，这些集成开发环境可帮助我们创建 HTML 文件的框架。

（2）页面设计。

在 Markets.html 页面中，加入如下代码：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>实践 10：费城农贸市场</title>
    <link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/ theme/default/style.css">
    <style>
        .smallmap {
            width: 512px;
            height: 512px;
            border: 1px solid #ccc;
        }

        #docs p {
            margin-bottom: 0.5em;
        }
    </style>
    <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
    </script>
    <script>
    </script>
</head>
<body onload="init()">
    <h1 id="title">费城农贸市场</h1>

    <div id="map" class="smallmap"></div>

    <div id="docs">
        <p>单击某一农贸市场，获取更多信息。</p>
    </div>
```

```
</body>
</html>
```

在上面的代码中，包含了 HTML 的 head 与 body 两大部分。虽然做了部分简化，但大部分的代码是从 OpenLayers 开发者实例中复制过来的。

在 head 部分，引用了运行在 CloudFlare 服务器上的 OpenLayers 的 JavaScript 文件与 CSS 样式文件。此外，还定义了两个样式。

在 body 部分，加入了 id 为 map 的 div，其样式的类设置为了 smallmap。该类在 head 部分的 style 中进行了定义，规定了地图的宽与高。

下面就需要在 head 的 script 中加入 JavaScript 代码，实现地图功能。

（3）定义全局变量。

在<script>与 </script>中，加入如下代码：

```
var fromProjection = new OpenLayers.Projection("EPSG:4326"); // WGS 1984
var toProjection = new OpenLayers.Projection("EPSG:900913"); // Web 墨卡托投影
var map;
function init() {
}
```

上述代码定义了 3 个全局变量，两个表示空间参照系的变量，一个是表示地图的变量。此外，还创建了初始化函数 init()的框架，我们需要在该函数中实现地图功能。

（4）在地图中加入切片图层。

在 init()函数中，加入如下代码：

```
map = new OpenLayers.Map("map", { projection: toProjection });

// 加入切片图层
var tiles = new OpenLayers.Layer.XYZ(
    "PhillyBasemap",
    [
        "http://localhost/PhillyBasemap/${z}/${x}/${y}.png"
    ],
    {
        attribution: "Data copyright OpenStreetMap contributors",
        sphericalMercator: true,
        wrapDateLine: true,
        numZoomLevels: 18
    }
);
map.addLayer(tiles);
```

上述代码首先创建了一个地图对象，并定义了其坐标系。然后创建了一个切片图层，指

向第 5 章实践 9 中创建的发布在 IIS 中的 PhillyBasemap 切片地图。

（5）在地图中加入 WMS 图层。

接着在 init()函数中加入如下代码：

```
// 加入 WMS 图层
var layer = new OpenLayers.Layer.WMS(
    "WMS", "http://localhost:8080/geoserver/webgis/wms",
    {
        LAYERS: 'webgis:FarmersMarkets', transparent: true
    },
    {
        singleTile: true,
        isBaseLayer: false
    }
);
map.addLayer(layer);
```

上述代码实现了将农贸市场 WMS 图层叠加在切片图层之上。将 singleTile 属性设置为 true，表示需要 WMS 服务将整个地图窗口中数据返回为一张图像，而不是多张填充地图窗口的正方形图像。

isBaseLayer 属性同样也很有用，它确保 WMS 图层处于正确的顺序，并且将其背景设置为透明。

（6）居中地图。

接着在 init()函数中加入如下代码：

```
// 居中地图
map.setCenter(new OpenLayers.LonLat(-75.145, 40).transform(fromProjection,toProjection), 11);
```

由于到城市街区尺度大概对应的是 20 级，因此如果没有上述代码，根本显示不了费城地图。上述代码将地图放大到 11 级，并将地图中心设置为费城的中心。

（7）实现交互功能。

仍然在 init()函数中加入如下代码：

```
// 实现单击查询功能
var info = new OpenLayers.Control.WMSGetFeatureInfo({
    url: 'http://localhost:8080/geoserver/webgis/wms',
    title: '单击查询要素',
    queryVisible: true,
    infoFormat: "application/json",
    eventListeners: {
        getfeatureinfo: function (event) {
            // 解析查询得到的响应
```

```
                var response = JSON.parse(event.text);
                if (response.features.length !== 0) {
                    var returnedFeature = response.features[0];
                    // 设置弹出窗口
                    map.addPopup(new OpenLayers.Popup.FramedCloud(
                      "农贸市场信息",
                      map.getLonLatFromPixel(event.xy),
                      null,
                      "<b>" + returnedFeature.properties.NAME + "</b><br />" +
                              returnedFeature.properties.ADDRESS,
                      null,
                      true
                    ));
                }
            }
        }
    });

    map.addControl(info);
    info.activate();
```

上述代码实现的是根据用户的单击位置查询到点击的要素，并将该要素的名称与地址显示在弹出窗口中。代码与“6.3.3 查询实例”介绍的查询类似。最大的不同是，在上述代码中指定要求从 WMS 返回 JSON 格式的响应，而不是 HTML 格式。JSON 格式的响应本身就是一个 JavaScript 对象，因此很容易得到各个字段的值。

（8）测试程序。

用 Firefox 直接打开 Markets.html 文件，得到图 6.7 所示的页面。

程序运行如果有错误，请参看下载文件“Codes\Walkthrough10”文件夹中的 Markets.html 文件。

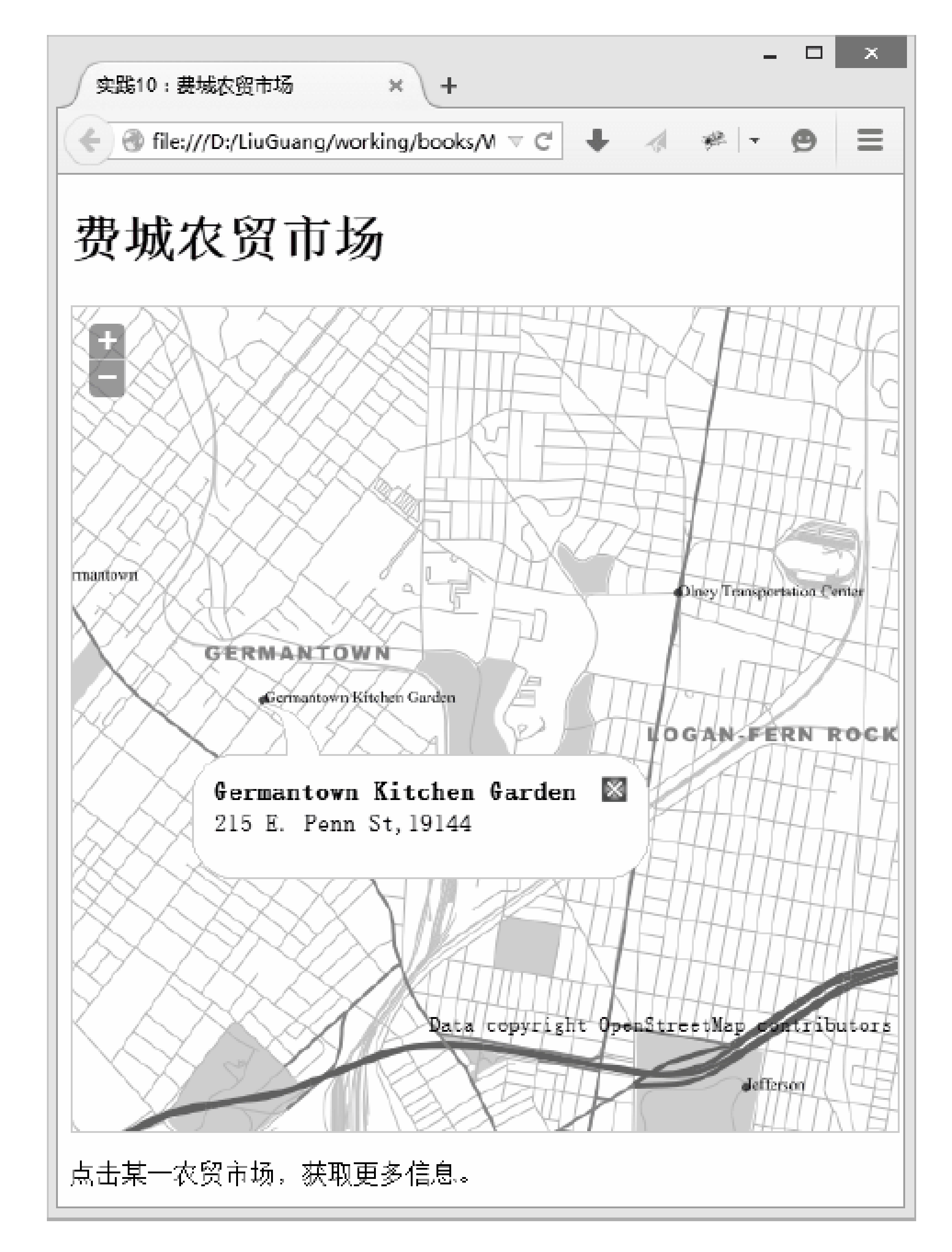


图 6.7　程序运行效果

6.5　习　题

（1）利用 OpenLayers，将第 4 章习题要求发布的 WMS 图层叠加到一切片地图上。该切片地图可以是第 5 章习题要求发布的切片，也可以是第三方的切片地图，例如 OpenStreetMap、必应地图等。要求实现查询 WMS 服务，将要素的详细信息显示在弹出窗口中。

（2）仔细阅读 GIS Stack Exchange 中一篇关于各 Web 地图 API 比较的讨论（http://gis.stackexchange.com/questions/8032/how-do-various-javascript-mapping-libraries-compare?rq=1），然后寻找分别使用 OpenLayers、Leaflet 与另一 Web 地图 API 构建的地图应用。分析其源代码，并编写一文档，说明如下内容：

- 地图应用的 URL。
- 地图应用使用了哪些 API。
- 在地图中包含了哪些服务与图层。
- 主要使用了 API 哪些类与功能。最好能给出这些类的帮助文档的链接地址。
- 通过分析源代码，学习了哪些编码技巧。
- 该地图应用还可以在哪些方面进行改进。

第 7 章

在客户端绘制矢量数据

从本章可以学习到：

- 在客户端绘制矢量数据的优势与挑战
- 使用 KML 矢量数据
- 使用 GeoJSON
- 在 OpenLayers 中符号化矢量图层
- 在 OpenLayers 使用 GeoJSON 图层
- 访问用户 KML 数据

在过去的几年里，我们亲眼目睹了在 Web 浏览器中令人难以置信的创新和进步。在现代浏览器中一系列全新的功能因 HTML5 而出现。在 HTML5 标准提供的许多功能中，提高 GIS 的关键在于 HTML5 的 Canvas。Canvas 基本上是在浏览器中动态生成的一个位图。

在上一章中，介绍了从 WMS 服务得到一张服务器端绘制好的图像，将其作为业务专题图层。该方式由于返回的是一张图像，虽然用户可以通过单击得到某要素更详细的信息，但是用户在与专题图层交互时，例如将鼠标移动到某要素上面，应用程序并不能知道并显示可选择该要素了，因此交互性并不强。本章将介绍在地图中显示专题图层的另一种方式，那就是将原始数据发送到客户端（例如一个网页浏览器），由客户端负责绘制。这是当前许多 Web GIS 普遍采用的方式，所有的复杂的符号系统和地图绘制功能将转移到客户端，使服务器只需要提供原始的矢量数据和属性数据。这意味着在地图引擎可以更有效地响应，以增强交互性以及提升性能。

本章将介绍两类可发送到浏览器中常用的矢量数据格式，分别是 KML 与 GeoJSON，并演示如何在 OpenLayers 中增加这两类图层。

7.1　在客户端绘制矢量数据的优势与挑战

前面介绍的都是从服务器上获取图像显示在地图中，不管是通过事先绘制的切片还是动态绘制的 WMS 地图。另一种方法是将包含要素空间坐标与属性信息的文本发送到客户端，然后由客户端负责绘制该图层。该方式如果使用得到，可明显提升 Web GIS 的速度与交互性。

虽然浏览器并没有 GIS 的概念，但是却也能绘制矢量图形。其实绘制矢量数据也不复杂，就是将屏幕坐标与符号连接的过程。Web 地图 API 一般都能从 GeoJSON 或 KML 文件中读取坐标值，然后转换为屏幕坐标，最后进行绘制。

7.1.1　客户端绘制矢量数据的优势

响应速度与交互性是在客户端绘制矢量数据的最主要两大优势。一旦从服务器获得了矢量数据，Web 地图用户与数据的交互就非常迅速，不会有任何延迟。假设有一个 Web 地图显示中国男子篮球职业联赛的所有球队，基础底图来自天地图的切片图层，篮球球队数据来自一个 GeoJSON 文件。当地图加载时，浏览器获取了 GeoJSON 文件，其包含每支球队的地理坐标以及所有其他属性信息。现在用户可以单击每支球队查看其信息，但不再需要向服务器发送请求。而对于前一章介绍的农贸市场混搭应用，每次单击某一农贸市场时，都需要向服务器发送一个 WMS 的 GetFeatureInfo 请求。

另外，假设应用程序需要实现用户每次将鼠标悬停在某球队上时，用高亮符号显示该球队，以便提示用户可以点击该符号。Web 浏览器可以用非常快的速度实现该效果。但是，如果每次悬停事件都需要向服务器发送请求，那么应用程序必定陷入瘫痪。

7.1.2 客户端绘制矢量数据的挑战

虽然客户端绘制矢量图形具有相应速度快与交互性高的优势，但是并不是所有情况都适合客户端绘制矢量。如果需要同时绘制成百上千个要素，或者绘制包含大量结点的多边形，那么更佳的选择可能是在服务器端绘制地图，然后将其发送到客户端。如果浏览器一次绘制的矢量图形太多，那么响应速度就会变得极其缓慢。此外，由于客户端必须下载所有的坐标，那么传输大量复杂图形也会导致网络堵塞。

为了保持良好的性能，最好针对每个比例尺，至少在小比例尺，将要在客户端绘制的图层数据尽可能地进行综合。例如，当在全国尺度下显示我国国界时，不需要使用一个包含福建省每一个很小的沿海岛屿的文件。只有放大到一定比例尺后才加载有更详细数据的文件。

此外，标注也是在浏览器端绘制图形的另一个挑战。虽然浏览器可以在屏幕上给定的坐标绘制文本，但是却没有像 GeoServer 与 TileMill 所拥有的强大的标注位置放置算法。结果可能是标注相互叠加。比较好的方式是让用户通过交互方式来发现标注，当用户单击某要素时，将标准显示在弹出窗口或 HTML div 中。

最后，通过 Web 浏览器所提供的符号的选择相对比较基本。可以在浏览器中绘制图形，如 SVG 文件，但是无法像 TileMill 或 ArcGIS 程序一样绘制复杂的线条和图形填充。当然，如果客户端恰好是一个桌面应用程序（如 QGIS），那么就不必过分担心可用符号的选择。

7.1.3 客户端如何绘制矢量数据

Web 地图 API 通常都提供了在浏览器端绘制矢量图层的类，但是不同的 API 使用的类名是不一样的。对于一些简单的独立的矢量要素，可能使用的是名为 marker 的类。而对于复杂的图层，可能使用的是 FeatureGroup（Leaflet）或 FeatureLayer（ESRI）类。OpenLayers 中的 Layer.Markers 与 Layer.Vector 类分别对应上述两个目的。

像 QGIS 这样的客户端应用程序可查看 KML、GeoJSON、GML 以及其他多种文本类型的矢量要素数据。

7.1.4 从服务器获取数据的方法

当在客户端定义了一矢量图层，那么需要指定客户端如何从服务器上获取数据。需要注意的是，客户端从服务器请求的不再是地图的图像，而是要获取矢量坐标以及相关的属性信息。每个 Web 地图 API 对从服务器获取矢量数据的方法都有自己的专业术语，在 OpenLayers 中称为策略。

从服务器获取矢量数据的一些主流方法主要有以下一些。

（1）在图层加载时获取所有的数据。在 OpenLayers 中，称这种方法为 Fixed 策略（http://dev.openlayers.org/docs/files/OpenLayers/Strategy/Fixed-js.html）。该方式在初始化时性能会有所损失，但是此后再也不需要向服务器发送其他请求，因此确保了应用程序随后的响

应速度。很明显，该方法对于非常大的数据量不适合。

（2）只获取当前地图视图范围内的数据。当地图视图改变时，再向服务器发送一新的请求。在 OpenLayers 中，该方式称为 BBOX 策略：http://dev.openlayers.org/docs/files/OpenLayers/Strategy/BBOX-js.html。

对于数据量大的矢量文件，一次将所有数据下载不太可能时，这种方式比较合适。但是当用户快速放大缩小或平移地图时，应用程序就响应不过来了。一种改进方式是考虑保留已经请求的要素数据。

（3）根据过滤或查询条件只从数据集中获取部分要素的矢量数据。在 OpenLayers 中，该方式称为 Filter 策略：http://dev.openlayers.org/docs/files/OpenLayers/Strategy/Filter-js.html。

该方式能缩小请求数据的范围，既避免了下载所有的数据，而又保留了 Fixed 策略的高响应效果。

此外，还有一些上述策略的改进版本。在 OpenLayers 中，Refresh 策略（http://dev.openlayers.org/ docs/files/ OpenLayers/Strategy/Refresh-js.html）在指定的时间间隔重新获取所有数据。如果矢量数据表示的是不断变化的现象，例如舰队、车队等，该方式就非常有用。

7.2　使用 KML 矢量数据

KML（Keyhole Markup Language，Keyhole 标记语言）由于可通过谷歌地球、谷歌地图和 ArcGIS Explorer 等许多免费应用程序进行查看，因此成为了 GIS 中矢量要素的一种非常受欢迎的格式。KML 是基于一个开放规范的表达地理标记的 XML 语言。该开放规范原来由谷歌维护，不过现已成为 OGC 标准大家庭中的一员。KML 可以由要素和栅格元素组成，这些元素包括点、线、面和影像，以及图形、图片、属性和 HTML 等相关内容。尽管通常将 ArcGIS 中的数据集视为独立的同类元素（例如，点要素类只能包含点，栅格只能包含像元或像素，而不能包含要素），但单个 KML 文件却可以包含不同类型的要素，并可包含影像。

7.2.1　KML 简介

KML 中最重要的 XML 标签是地标（placemark），它定义了一些地理要素、一些符号以及其他一些可显示在弹出窗口中的额外信息。读者可下载 http://dev.openlayers.org/releases/OpenLayers-2.13.1/ examples/kml/sundials.kml 文件，并用文本编辑器打开。在其中可以看到有许多地标标签，例如：

```
<Placemark>
    <name>Sundial, Plymouth, Devon, UK</name>
    <description><![CDATA[The gnonom is 27 foot high, the pool has 21 feet diameter.  It was designed by architect Carole Vincent from Boscastle in Cornwall and was unvieled by Her Majesty the Queen on Friday July 22nd 1988 for a cost of cost £70,000 . The sundial runs one hour and seventeen minutes behind local clocks.
        <img src="http://www.photoready.co.uk/people-life/images/sundial-fountain.jpg">
```

```
            Image source:<a href="www.photoready.co.uk</a>]]>
        </description>
        <LookAt>
            <longitude>-4.142398271107962</longitude>
            <latitude>50.37145390235462</latitude>
            <altitude>0</altitude>
            <range>63.33410419881957</range>
            <tilt>0</tilt>
            <heading>-0.0001034131369701296</heading>
        </LookAt>
        <styleUrl>#msn_sunny_copy69</styleUrl>
        <Point>
            <coordinates>-4.142446411782089,50.37160252809223,0</coordinates>
        </Point>
    </Placemark>
```

这个特定的地标只有一个坐标，包含在 Point 标签中。对于线与多边形要素，分别使用的是 LineString 与 Polygon 标签。

在 Description 标签中包含的是 HTML，这非常适合在弹出窗口中显示。

7.2.2 在 OpenLayers 中使用 KML

一个完整的 KML 文件相对都比较长，不过并不需要我们从头解析 KML 文件。当前大多数的 Web 地图 API 都提供了相关类来访问 KML 文件。在 OpenLayers 中，对应的是 OpenLayers.Layer.Vector 类，调用该类时只需要提供 KML 文件所在的路径，解析 KML 文件、显示内容等其他工作全部由 OpenLayers 完成。

http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/sundials.html 连接展示了如何使用 OpenLayers.Layer.Vector 类，将一个 KML 文件作为一个图层加入到地图中。该实例主要代码如下：

```
var sundials = new OpenLayers.Layer.Vector("KML", {
    projection: map.displayProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "kml/sundials.kml",
        format: new OpenLayers.Format.KML({
            extractStyles: true,
            extractAttributes: true
        })
    })
});
```

```
map.addLayers([wms, sundials]);
```

在上面的代码中，将策略设置为了 Fixed，这就意味着当地图加载时，将 KML 文件中的所有要素加载到浏览器中。

KML 加载使用的是 OpenLayers.Protocol.HTTP，即 HTTP 协议，与其对应的是 OpenLayers.Protocol.SQL，后者用于连接数据库。在使用 OpenLayers. Protocol.HTTP 访问 KML 时，要指定要访问数据的格式为 OpenLayers.Format.KML，以及指定 KML 文件的路径。如果查看 OpenLayers.Format（http://dev.openlayers.org/docs/files/OpenLayers/Format-js.html）的帮助文档，可以看出可以在地图中加载很多格式的矢量数据。

7.3　使用 GeoJSON

GeoJSON 也是一个在 Web 地图中显示矢量数据的广泛使用的数据格式。其主要特点是基于 Javascript 对象表示法。在 GeoJSON 中，一个矢量要素及其属性使用一个 JavaScript 对象来表示，这样就非常方便解析其几何图形与字段。

7.3.1　GeoJSON 简介

对于同样的要素，相对于 KML 这种基于 XML 结构的格式，GeoJSON 要小许多。但是 GeoJSON 并不像 KML 那样总是包含样式信息。需要在客户端定义样式，这意味着需要编写一些 JavaScript 代码或使用 OpenLayers 中的默认样式。

由于 GeoJSON 简单而且加载速度快，因此越来越多的开发者更愿意使用 GeoJSON。

GeoJSON 可以描述的对象包括几何图形、要素和要素集。几何图形的类型有点、线、面、多点、多线、多面与几何图形集合。要素包含了几何图形信息以及附加的一些属性信息。要素集即为要素的集合。

下面展示的是一包含要素集的 GeoJSON。该要素集中只包含了一个要素(美国蒙大拿州)，但可以很容易扩展以包含其他要素。该 GeoJSON 的主体是定义该州边界线的顶点，但是也包含了 fips、name 等少数几个属性。


```
{"type":"FeatureCollection","features":[{"type":"Feature","id":"USA-MT","properties":{"fips":"30","name":"
Montana"},"geometry":{"type":"Polygon","coordinates":[[[-104.047534,49.000239],[-104.042057,47.861036],[-104
.047534,45.944106],[-104.042057,44.996596],[-104.058488,44.996596],[-105.91517,45.002073],[-109.080842,45.0
02073],[-111.05254,45.002073],[-111.047063,44.476286],[-111.227803,44.580348],[-111.386634,44.75561],[-111.
616665,44.547487],[-111.819312,44.509148],[-111.868605,44.563917],[-112.104113,44.520102],[-112.241036,44.
569394],[-112.471068,44.481763],[-112.783254,44.48724],[-112.887315,44.394132],[-113.002331,44.448902],[-11
3.133778,44.772041],[-113.341901,44.782995],[-113.456917,44.865149],[-113.45144,45.056842],[-113.571933,45.
128042],[-113.736241,45.330689],[-113.834826,45.522382],[-113.807441,45.604536],[-113.98818,45.703121],[-11
4.086765,45.593582],[-114.333228,45.456659],[-114.546828,45.560721],[-114.497536,45.670259],[-114.568736,4
```

```
5.774321],[-114.387997,45.88386],[-114.492059,46.037214],[-114.464674,46.272723],[-114.322274,46.645155],[-
114.612552,46.639678],[-114.623506,46.705401],[-114.886399,46.809463],[-114.930214,46.919002],[-115.302646
,47.187372],[-115.324554,47.258572],[-115.527201,47.302388],[-115.718894,47.42288],[-115.724371,47.696727],[
-116.04751,47.976051],[-116.04751,49.000239],[-111.50165,48.994762],[-109.453274,49.000239],[-104.047534,49
.000239]]]}}]}
```


在上面的 GeoJSON 中，该 JavaScript 对象包含了好几个小的 JavaScript 对象。最底层是一个多边形对象，该多边形对象包含在一个要素对象中，而该要素对象是一个要素集对象的一部分。GeoJSON 规范（http://geojson.org/geojson-spec.html#introduction）给出了这些对象如何组织的详细描述。虽然我们很少直接读写 GeoJSON 文件，但是熟悉其中的结构还是非常有必要的。

7.3.2　在 OpenLayers 中使用 GeoJSON

虽然可以在 JavaScript 代码文件中包含 GeoJSON，但是为了使用与维护简便，通常将一个 GeoJSON 放在一个单独的文件中。然后在代码中引用该文件。下面是一段使用 OpenLayers 访问 GeoJSON 的代码：

```
var vector = new OpenLayers.Layer.Vector("GeoJSON", {
    projection: "EPSG:4326",
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "geojson-reprojected.json",
        format: new OpenLayers.Format.GeoJSON()
    })
});
```

上面的代码与访问 KML 的代码非常类似，仅有的区别是 OpenLayers.Protocol.HTTP 中的 url 与 format。url 属性指向 JSON 文件，可以是完整的 URL 或相对路径的 URL。为了简单，这里省略了将该图层加入到地图中的代码行。

当使用任何 Web 地图 API 创建 GeoJSON 图层时，要注意 API 是如何要求定义、组织与引用 GeoJSON 文件的。在 OpenLayers 中可以引用一个纯粹的 GeoJSON，但是在 Leaflet 中则需要将 GeoJSON 定义为一个 JavaScript 对象，如下所示：

```
var <yourVariableName> = <yourGeoJSON>;
```

在 QGIS 中，可将任何的矢量图层保存为 GeoJSON，而且大多数 Web 地图 API 都为 GeoJSON 提供非常方便使用的类，将其显示为矢量格式。在商业软件领域中，ESRI 对 GeoJSON 的支持不怎么积极，反而推出了其自身的基于 JSON 的几何图形格式（http://resources.arcgis.com/en/help/arcgis-rest-api/02r3/02r3000000n1000000.htm），包含在其 GeoServices REST 规范与 ArcGIS REST API 中。不过，ESRI 也已非正式地共享一个开源的

JavaScript 库，以便进行两种格式之间进行转换。

GeoJSON 规范还不是一个 OGC 的规范。当前，OGC 只发布了基于 XML 的 GML 规范，明显还缺乏一个基于 JSON 的定义矢量 GIS 对象的规范。这种缺乏一个 OGC 认可的 JSON 规范的现象，引起了 FOSS 社区在 2013 辩论 OGC 是否应该采用 ESRI 的生成地理服务 REST 规范。该规范将给予 OGC 一个基于 JSON 的 GIS 数据格式，但有些人担心该格式与一家商业软件公司有关联，而对此反对。

7.4　在 OpenLayers 中符号化矢量图层

当在浏览器中获得了矢量数据之后，需要定义绘制要素的符号。对于 GeoJSON 这类不包含样式的数据，不可能像从服务器请求图像那样得到事先准备好的样式。如果不为矢量图形定义符号，那么一般就是用 API 默认的符号。

可以直接在矢量图层的构造函数中定义样式。下面的代码演示了创建一个 GeoJSON 图层，该图层显示城市公园（多边形）。在构造函数中通过直接定义一个 JSON 对象（加粗文字）来表示样式：

```
var gardensLayer = new OpenLayers.Layer.Vector("Gardens", {
    projection: fromProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "gardens.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    style: {
        'strokeWidth': 4,
        'fillColor': '#ff00ff',
        'strokeColor': '#B04173'
    }
});
```

在上面的代码中，规定了画笔的宽度、填充颜色以及画笔的颜色。颜色使用的是十六进制表示法。上述代码指定的是带紫色边线品红色填充的多边形符号，如图 7.1 所示。在编写代码时，可使用在线颜色拾取工具（http://www.colorpicker.com/）来得到满足需求颜色的十六进制值。



图 7.1　在矢量图层类中直接指定样式

在浏览器中绘制矢量图层的一个优点是，可以快速改变样式以响应某些事件。例如，当用户将鼠标悬停在某要素上时，可以改变该要素的符号的颜色。在 OpenLayers 中，可以使用 OpenLayers.Style 定义不同的样式对象，并把它们连接到地图事件上。

OpenLayers.StyleMap 是一个很特别的对象，它包含多种样式以及每种样式使用的规则。下面的代码演示了当某一公园要素被单击时，该被选择的要素变为蓝色填充。

首先，使用 OpenLayers.Style 类定义非选择与选择状况下的两个样式对象。

```
// 创建样式
// 品红色填充样式
var gardenStyle = new OpenLayers.Style({
    'strokeWidth':4,
    'fillColor':'#ff00ff',
    'strokeColor':'#B04173'
});

// 蓝色填充样式
var selectedGardenStyle = new OpenLayers.Style({
    'strokeWidth':4,
    'fillColor':'#00fffb',
    'strokeColor':'#0000ff'
});
```

然后，使用 OpenLayers.StyleMap 定义默认样式与选择状体样式。

```
// 指定公园使用的默认样式与选择状态样式
var gardenStyleMap = new OpenLayers.StyleMap({'default': gardenStyle,'select': selectedGardenStyle});
```

接着在初始化 GeoJSON 图层时，在构造函数中引用该 StyleMap 对象（加粗文字）。

```
// 定义公园 GeoJSON 图层
var gardensLayer = new OpenLayers.Layer.Vector("Gardens", {
    projection: fromProjection,
```

```
        strategies: [new OpenLayers.Strategy.Fixed()],
        protocol: new OpenLayers.Protocol.HTTP({
            url: "gardens.geojson",
            format: new OpenLayers.Format.GeoJSON()
        }),
        styleMap: gardenStyleMap
    });
```

最后，在地图中加入一个 OpenLayers.Control.SelectFeature 控件。该对象监听单击事件，并根据样式映射自动对选择状态与非选择状态的要素应用对应的样式。

```
    // 监听要素被单击事件
    selectControl = new OpenLayers.Control.SelectFeature(gardensLayer);
    map.addControl(selectControl);
    selectControl.activate();
```

通过上面的代码，实现了当用户单击某公园时，用蓝色填充高亮表示。当用户单击了其他要素或在其他地方单击时，原来被选择的要素又返回原来的品红色填充样式。

7.5　实践 11：在 OpenLayers 使用 GeoJSON 图层

该实践将在费城切片地图上叠加分别代表城市公园与食品店的两种 GeoJSON 图层。用户可以通过单击某公园或食品店，查看该要素的名称。被单击的要素将改变颜色以表示被选中。

（1）准备使用的数据。

新建一个名为 Walkthrough11 的文件夹，将下载文件的“Data\Walkthrough11”文件夹中的 4 个文件复制到新建的 Walkthrough11 文件夹中。其中 gardens.geojson 与 pantries.geojson 分别是公园与食品店矢量数据，另外的 pantries.svg 与 pantries_selected.svg 是两个用于符号化食品店的 SVG 文件。其中黄色符号用于非选择状态的要素，而蓝色符号用于被选择要素。

对于.geojson 的数据，可以通过 QGIS 导出矢量数据来得到。而且“C:\Program Files\QGIS Brighton\apps\qgis\svg”文件夹中包含了 QGIS 中使用的图标。

（2）设计 HTML 页面。

在 Walkthrough11 文件夹中增加一个名为 UseGeoJSON.html 的文件。其代码如下：

```
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="utf-8">
        <meta  name="viewport"  content="width=device-width,  initial-scale=1.0,  maximum-scale=1.0,
user-scalable=0">
        <meta name="apple-mobile-web-app-capable" content="yes">
```

```
        <title>实践 11：费城公园与食品店</title>
        <link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/theme/default/style.css">
        <style>
            .smallmap {
                width: 512px;
                height: 512px;
                border: 1px solid #ccc;
            }

            #summaryLabel p {
                margin-bottom: 0.5em;
            }
        </style>
        <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
</script>
        <script>
        </script>
</head>
<body onload="init()">
        <h1 id="title">社区公园与食品店</h1>
         <div id="map" class="smallmap"></div>
         <div id="summaryLabel">
             <p>单击地图中的公园或食品店获取更多信息</p>
         </div>
</body>
</html>
```

（3）定义投影与地图对象。

在<script>与 </script>之间加入如下代码，定义投影与地图对象。

```
var fromProjection = new OpenLayers.Projection("EPSG:4326"); // WGS 1984
var toProjection = new OpenLayers.Projection("EPSG:900913"); // Web 墨卡托投影
var map;
function init() {
    map = new OpenLayers.Map("map", { projection: toProjection });
}
```

（4）定义费城切片地图图层。

在 init()加入如下代码，定义费城切片地图图层。

```
// 定义费城切片地图图层
var tiles = new OpenLayers.Layer.XYZ(
    "PhillyBasemap",
```

```
        [
            "http://localhost/PhillyBasemap/${z}/${x}/${y}.png"
        ],
        {
            attribution: "Data copyright OpenStreetMap contributors",
            sphericalMercator: true,
            wrapDateLine: true,
            numZoomLevels: 18
        }
);
```

（5）定义公园矢量图层。

接着在 init()加入如下代码，定义公园矢量图层。

```
// 品红色填充样式，用于未被选择的公园要素
var gardenStyle = new OpenLayers.Style({
    'strokeWidth': 4,
    'fillColor': '#ff00ff',
    'strokeColor': '#B04173'
});

// 蓝色填充样式，用于被选择公园要素
var selectedGardenStyle = new OpenLayers.Style({
    'strokeWidth': 4,
    'fillColor': '#00fffb',
    'strokeColor': '#0000ff'
});

// 为公园图层定义样式映射
var gardenStyleMap = new OpenLayers.StyleMap({ 'default': gardenStyle, 'select': selectedGardenStyle });

// 定义公园 GeoJSON 图层
var gardensLayer = new OpenLayers.Layer.Vector("Gardens", {
    projection: fromProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "gardens.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: gardenStyleMap
});
```

（6）定义食品店矢量图层。

接着在 init()加入如下代码，定义食品店矢量图层。

```
// 黄色符号用于未选择的食品店要素
var pantryStyle = new OpenLayers.Style({
    externalGraphic: 'pantries.svg',
    graphicWidth: 25,
    graphicHeight: 25,
    graphicYOffset: 0
});

// 蓝色符号用于被选择的食品店要素
var selectedPantryStyle = new OpenLayers.Style({
    externalGraphic: 'pantries selected.svg',
    graphicWidth: 25,
    graphicHeight: 25,
    graphicYOffset: 0
});

// 为食品店图层定义样式映射
var pantryStyleMap = new OpenLayers.StyleMap({ 'default': pantryStyle, 'select': selectedPantryStyle });

// 定义食品店 GeoJSON 图层
var pantriesLayer = new OpenLayers.Layer.Vector("Pantries", {
    projection: fromProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "pantries.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: pantryStyleMap
});
```

上面的代码演示了如何使用.svg 文件作为符号。由于食品店是点图层，因此不需要设置画笔与填充颜色。

（7）在地图中加入图层。

接着在 init()加入如下代码，用于实现在地图中加入上面定义的 3 个图层，并将地图调整到合适的比例级别。

```
map.addLayers([tiles, gardensLayer, pantriesLayer]);
map.setCenter(new OpenLayers.LonLat(-75.15, 40).transform(fromProjection, toProjection), 12);
```

在上面的代码中，演示了如何使用数组，一次将 3 个图层加入到地图中。

（8）加入事件监听控件。

接着在 init()加入如下代码，加入监听要素选择与取消选择事件的控件。

```
// 监听要素选择与取消选择事件
var selectControl = new OpenLayers.Control.SelectFeature([gardensLayer, pantriesLayer], {
    onSelect: onFeatureSelect,
    onUnselect: onFeatureUnselect
});

map.addControl(selectControl);
selectControl.activate();
```

上面的代码将一个新的 SelectFeature 控件加入到了地图中，该控件用于监听要素的单击事件。在 SelectFeature 控件的构造函数中，通过第一个参数，同时将公园图层、食品店图层与该控件进行了绑定。

此外，在定义 SelectFeature 控件的构造函数中，还引用了两个函数，onFeatureSelect 与 onFeatureUnselect，分别用于处理选中与取消选中事件。在下一步需要实现这两个函数。

（9）加入要素选中与取消选中事件处理函数。

在 init()加入如下代码，处理要素选中与取消选中事件。

```
// 处理要素选中事件
function onFeatureSelect(feature) {
    var featureName = feature.attributes.name || "无名称要素";
    // 插入一段 HTML，显示要素的名称
    document.getElementById('summaryLabel').innerHTML = '<p style="font-size:18px"><b>' +
featureName + '</b></p>';
}

// 处理要素取消选中事件
function onFeatureUnselect(feature) {
    // 将 HTML 返回到原始状态
    document.getElementById('summaryLabel').innerHTML = '<p>单击地图中的公园或食品店获取更多
信息</p>';
}
```

上述两个函数是由 SelectFeature 控件触发的事件处理函数。当用户单击某要素时，触发 onFeatureSelect 函数。该函数首先获取被选择的 GeoJSON 要素的 name 属性，如果没找到，则将其设置为“无名称要素”，然后将名为 summaryLabel 的 div 中的内容替换为要素的名称。

当用户取消选择某要素时将触发 onFeatureUnselect 函数，该函数将名为 summaryLabel 的 div 中的内容设置为初始值。

（10）运行与调试程序。

用 Firefox 等浏览器直接打开 UseGeoJSON.html 文件，单击公园与食品店要素，查看是否

改变符号，以及是否在地图下方显示该要素的名称。程序运行效果如图 7.2 所示。程序运行如果有错误，请参看下载文件“Codes\Walkthrough11”文件夹中的 UseGeoJSON.html 文件。

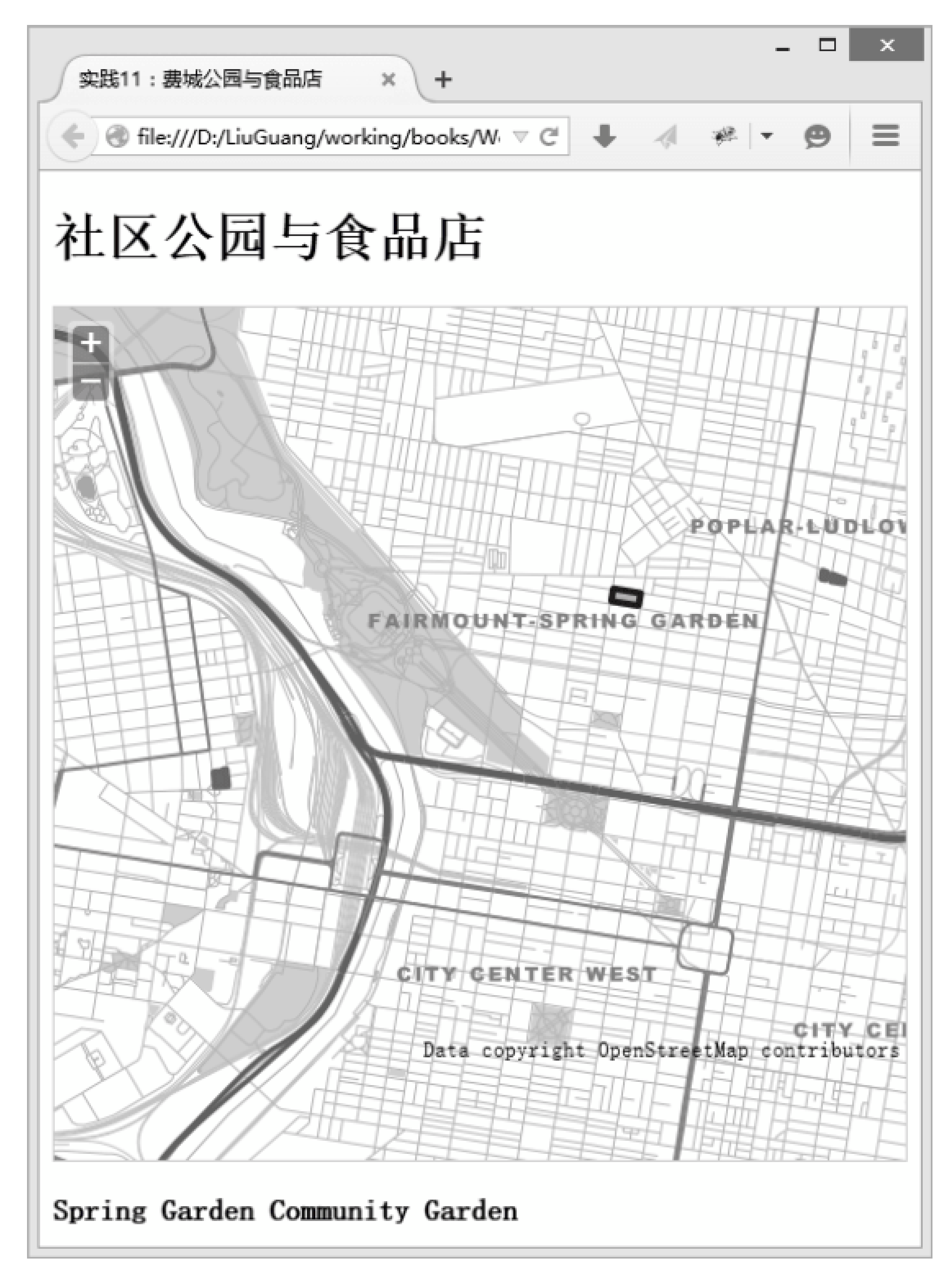


图 7.2　程序运行效果

7.6　实践 12：访问用户 KML 数据

在上一个实践中，演示了如何访问服务器端的 GeoJSON 数据，在本实践中将介绍 KML 数据的访问。此外，本实践还将演示如何访问用户自己的数据，而不是服务器端的数据。该功能的实现得益于 HTML5 中的文件 API。当用户拖入本地的 KML 文件后，地图将显示该文件包含的矢量要素，用户单击要素时，将弹出一个小窗口以显示 KML 文件中该要素的 description 字段内容。

7.6.1　页面设计

新建一个名为 Walkthrough12 的文件夹，在其中新建名为 UserFiles.html 的文件。在其中加入如下代码：

```
<!DOCTYPE html>
```

```
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>实践 12：访问用户数据</title>
    <link rel="stylesheet" href="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/theme/default/style.css">
    <style>
        html, body {
            height: 100%;
        }
        #map {
            width: 100%;
            height: 80%;
            border: 1px solid black;
        }
        .olPopup p { margin:0px; font-size: .9em;}
        .olPopup h2 { font-size:1.2em; }
    </style>
    <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
</script>
    <script>
    </script>
</head>
<body onload="init()">
    <h1 id="title">使用 HTML5 File API 访问用户自己的数据</h1>

    <p id="shortdesc">
        将 KML 文件拖放到地图中。
    </p>

    <div id="map"></div>
</body>
</html>
```

7.6.2　功能实现

（1）在地图中加入图层。

在<script>与 </script>之间加入如下代码，定义投影、地图对象与两个图层对象，以及将图层加入到地图中，并设置到恰当的显示级别与范围。

```
var WGS84 = new OpenLayers.Projection("EPSG:4326"); // WGS 1984
var mercator = new OpenLayers.Projection("EPSG:900913"); // Web 墨卡托投影
var map;
function init() {
    map = new OpenLayers.Map("map");

    // 使用 OpenStreetMap 作为基础底图
    var baseLayer = new OpenLayers.Layer.OSM("");

    var viewLayer = new OpenLayers.Layer.Vector("", {
        projection: WGS84,
        styleMap: new OpenLayers.StyleMap({
            fillColor: "blue",
            strokeColor: "blue",
            strokeWidth: 3
        })
    });
    map.addLayers([baseLayer, viewLayer]);
    var lonlat = new OpenLayers.LonLat(0, 0).transform(WGS84, mercator);
    map.setCenter(lonlat, 3);
}
```

当前 viewLayer 是一个空图层，当用户拖入文件后，再填充其数据内容。

（2）判断浏览器是否支持 HTML5 文件 API。

在浏览器中读取本地文件是 HTML5 的最大亮点之一。不过有一些老版本的浏览器并不支持 HTML5 的文件 API，因此首先需要判断浏览器是否具有该功能。

继续在 init()函数中加入如下代码，用于判断浏览器是否支持 HTML5 文件 API。

```
var shortdesc = document.getElementById('shortdesc');
if (!window.FileList || !window.FileReader) {
    shortdesc.textContent = '你的浏览器不支持本地文件 API。';
    return;
}
```

（3）实现文件拖放功能。

文件拖放也是 HTML5 的一大亮点，可以直接将本地电脑上的文件拖拽到网页中。这种直接拖放文件至浏览器的做法让传统的表单或者 Flash 上传黯然失色，极大提升了用户体验。

要实现文件拖放，首先要确定文件拖放区域，对于本实践来说最佳选择当然就是地图。然后就是设置其监听文件拖放事件。必须要绑定的事件有 dragover 和 drop，其他的都可以不绑定。dragover 和 drop 事件的处理函数内必须调用事件的 preventDefault() 函数，否则浏览器会进行默认处理，比如对于文本类型的文件会直接打开，非文本的文件可能会弹出一个下载文件框。

继续在 init()函数中加入如下代码，以实现文件拖放功能。

```
var fileDrop = document.getElementById('map');
fileDrop.addEventListener("dragenter", function (evt) {
    evt.stopPropagation();
    evt.preventDefault();
}, false);

fileDrop.addEventListener("dragover", function (evt) {
    evt.stopPropagation();
    evt.preventDefault();
}, false);

fileDrop.addEventListener("drop", function (evt) {
    evt.stopPropagation();
    evt.preventDefault();
    if (evt.dataTransfer.files[0]) {
        handleFile(evt.dataTransfer.files[0]);
    }
}, false);
```

（4）利用文件 API 读用户拖入的文件。

通过拖放事件的 dataTransfer 可获取数据传递对象，该对象中最重要的是 files 属性，它是用户拖放进浏览器的文件列表，是一个 FileList 对象，有 length 属性，可以通过下标访问其中某个文件。

上述代码指定了使用 handleFile 函数来处理用户拖入的文件。那么在该函数中需要读入 KML 数据，并将数据加入到 viewLayer 图层对象中。

继续在 init()函数中加入如下代码，实现解析文件并在 viewLayer 图层中加入要素数据。

```
var handleFile = function (file) {
    var reader = new FileReader();
    reader.onload = function (evt) {
        if (evt.error) {
            readerror();
            return;
        }

        var results = null;
        var content = evt.target.result;
        var engine = new OpenLayers.Format['KML']({
            internalProjection: mercator,
            externalProjection: WGS84,
```

```
                extractStyles: true
            });

            try {
                results = engine.read(content);
            } catch (e) {
            }

            if (!results || !results.length) {
                readerror();
                return;
            }
            viewLayer.destroyFeatures();
            viewLayer.addFeatures(results);
            map.zoomToExtent(viewLayer.getDataExtent());
        }
        reader.readAsText(file);
    }

    function readerror() {
        shortdesc.textContent = "拖入的文件不是标准的 KML 文件。";
    }
```

要读取文件，必须使用 FileReader 接口，该接口提供了读取文件的方法以及文件读取后的事件模型。

在上面的代码中，首先新建了一个 FileReader 对象，然后调用了其 readAsText()方法，按照文本文件的方式读取文件，文件读取完成后将触发 onload 事件。因此第三步是该事件的处理函数，在该函数中利用事件对象的 target.result 获得文件内容，接着利用 OpenLayers.Format 将文件内容解析为要素几何对象，并加入到 viewLayer 图层中。

（5）查看要素描述内容。

KML 文件中每个要素一般都包含一描述信息，可显示在弹出窗口中。要实现该功能，便需要处理 viewLayer 图层的要素选择与取消选择事件。

继续在 init()函数中加入如下代码，以实现查看要素描述内容。

```
    // 监听要素选择与取消选择事件
    var selectControl = new OpenLayers.Control.SelectFeature(viewLayer, {
        onSelect: onFeatureSelect,
        onUnselect: onFeatureUnselect
    });

    map.addControl(selectControl);
```

```
selectControl.activate();

// 处理要素选中事件
function onFeatureSelect(feature) {
    var content = "<h2>" + feature.attributes.name + "</h2>" + feature.attributes.description;
    if (content.search("<script") != -1) {
        content = "内容中包含 Javascript 代码！将跳过这些内容。<br>" + content.replace(/</g, "&lt;");
    }
    popup = new OpenLayers.Popup.FramedCloud("chicken",
        feature.geometry.getBounds().getCenterLonLat(),
        new OpenLayers.Size(100, 100),
        content,
        null, true, onPopupClose);
    feature.popup = popup;
    map.addPopup(popup);
}

// 处理要素取消选中事件
function onFeatureUnselect(feature) {
    if (feature.popup) {
        map.removePopup(feature.popup);
        feature.popup.destroy();
        delete feature.popup;
    }
}

function onPopupClose(evt) {
    selectControl.unselectAll();
}
```

（6）程序运行与调试。

用 Firefox 等浏览器直接打开 UserFiles.html 文件，从资源管理器或桌面中将一个 KML 文件拖入到地图中，并可显示该文件中包含的矢量数据。程序运行效果如图 7.3 所示，拖入的是下载文件“Data\KML”文件夹中的 sundials.kml 文件。程序运行如果有错误，请参看下载文件“Codes\Walkthrough12”文件夹中的 UserFiles.html 文件。



图 7.3　程序运行效果

7.7　习　题

（1）编写一个 Web GIS 应用程序，实现如下功能：

- 使用第 5 章习题指定创建的切片地图或 OpenStreetMap 作为基础底图。
- 从互联网上或自己创建矢量图层叠加在基础底图上。矢量图层的格式是 KML 或 GeoJSON。可使用谷歌地球或谷歌地图创建 KML，可使用 QGIS 创建 GeoJSON。
- 实现用户点击要素时，使用不同的符号显示该要素，并显示其详细信息。

（2）利用 HTML5 文件 API，编写一个 Web GIS 应用程序，实现如下功能：

- 使用 OpenStreetMap 作为基础底图。
- 支持用户将本地 KML、GPX 与 OSM 文件拖入到地图中，显示其中的矢量数据，并支持通过单击显示被选要素的详细属性信息。

第 8 章

主流 JavaScript 框架的使用与专题制图

从本章可以学习到：

- 主流 JavaScript 框架
- OpenLayers 的控件
- 基于属性值符号化图层
- 使用 OpenLayers 与 Dojo 进行专题制图

在前面的内容中介绍了如何创建不同类型的图层，并使用 OpenLayers 将它们叠加在一张地图上。在大多数情况下，对于一些简单的 Web GIS 应用只需要在基础底图上叠加一些专题点，以及少数几个弹出窗口就满足用户的需求量。事实上，对于许多从未将数据以地理信息方式展现的用户，当看到自己的数据呈现在这些基本 Web 地图上时就已经很兴奋了。不过，随着 Web GIS 的大众化，人们希望地图图层包含更多信息以及交互性更强。

本章将介绍引入主流 JavaScript 框架，增强 Web 地图的用户体验。并通过专题制图，以更丰富的形式展现空间信息。

8.1 主流 JavaScript 框架

随着 Web 2.0 的发展，在不是很影响性能的情况下，开发者都习惯把能用浏览器做的事情都由浏览器来完成，以减轻服务器的压力和带宽费用等，否则浏览器端所承载的工作越来越大。因此 JavaScript 已经成为了 Web 开发最基本的要求之一。而在现实的敏捷开发中，我们通常会选择一个 JavaScript 框架来取代烦琐的从头编写。这样会节省很多的时间，写代码也很清晰、便捷。

当前流行的开源 JavaScript 框架有 Prototype、jQuery、Mootools、Dojo 与 Ext JS 等。具体可以从以下几个方面来选择使用哪个框架：

（1）项目需求（即需要哪些特性，例如是否要求做出精美的界面、特效或其他功能）；

（2）是否支持 A 等级的浏览器（IE、Firefox 等）；

（3）文档的质量：是否完善（包含教程、API 和代码示例等）；

（4）框架的可扩展性如何？为框架写插件容易吗？

（5）是否喜欢它的 API 的风格？

（6）能从多大程度上统一 JavaScript 代码的风格？

（7）框架大小（太大的框架会导致用户下载时间的延长）；

（8）框架是否强迫改变写 HTML 的方式（Dojo 就是这样）？

（9）代码执行速度：性能如何？

（10）代码是否为模块化（Mootools 为高度模块化）？代码可重用性如何？

8.1.1 jQuery

jQuery 的主页为 http://www.jquery.com/。jQuery 的优点包括：

（1）简洁的设计思想：几乎所有操作都是以选择 DOM 元素（有强大的 Selector）开始，然后是对其操作（Chaining 等特性）。

（2）容易小，压缩后代码只有 20K 多一些（无压缩代码 94K）。

（3）Selector 和 DOM 操作的方便。

（4）链式语法：总是返回一个 jQuery 对象，可以连续操作。

例如如下代码：

```
$("p.surprise").addClass("ohmy").show("slow");
```

相当于首先查找 HTML 的<p>标签，且其 class 为“surprise”的 DHTML DOM 对象；然后将其 Class 属性多加上一个“ohmy”（通常是配 CSS 的定义做显示时的配色修改）；最后打开显示。

（5）文档的完整与易用性（每个 API 都有完整的例子，这是其他框架现在不能比的），而且在互联网上还有很多其他的文档和书籍。

（6）广泛的应用，包括 Google 代码也使用了 jQuery。

（7）简洁和简短的语法，容易记忆。

（8）可扩展性：有大量用户开发的插件可供使用（http://jquery.com/plugins/）。

（9）jQuery 的用户界面库（http://jquery.com/plugins/，基于 jQuery，但其与核心的 jQuery 是独立的）一直在不断发展中，包括拖放、缩放、对话框、标签页等多个组件。

（10）事件处理有很多方便的方法，如 click，而不是单一的 addEvent 之类的。

但是正由于设计思想是追求高效和简洁，没有面向对象的扩展，因此在设计思路上与 Mootools 不一样。

8.1.2　Mootools

Mootools 的主页为 http:// www.mootools.net/。Mootools 在设计上采用了面向对象的设计思想。Mootools 跟 Prototype 相类似，语法几乎一样。但它提供的功能要比 Prototype 多，而且更强大。

其主要优点包括如下一些：

（1）模块化，而且各模块代码非常独立，最小的核心只有 8K，最大的优点是可选择使用哪些模块，在使用的时候只导入使用的模块即可，完整的也不到 180K（没有压缩），压缩后不到 70K。

（2）语法简洁、直观。

（3）特效：这一点比 jQuery 稍强，现在也正在开发 Mootools 用户界面库。

（4）代码易阅读与修改。

（5）文档完整（最新的 1.2beta 的文档比以前更详细）。

Mootools 的缺点是修改了低层的一些类，如 Array 和 String 等，这也是设计思想的不同。此外，在 DOM 和 CSS 选择器也不如 jQuery 强大。

8.1.3　Ext JS

Ext JS 的主页为 http:// www.extjs.com/。Ext JS 在设计上采用了组件化思想，推进了互联网应用（Rich Internet Application，RIA）。

Ext JS 主要包含如下一些优点：

（1）强大的用户界面库，而且性能不错，这是其最大的优点。

（2）不管是用户界面库还是其他模块，响应速度都比较快。

（3）100%面向对象和组件化的思想，使用一致的语法和全局的命名空间。

（4）文档完整、规范、方便。

（5）活跃的社区，迅速增加的用户量。

（6）模块化实现，可扩展性强。

（7）所有的小部件都可直接使用，而不需要进行设置（当然，用户可以选择重新配置）。

Ext JS 的最大缺点是稍复杂，是以重量级的框架（包含大量 UI），体积大。如果导入 ext-all.js，压缩后也有近 500K。

8.1.4 Dojo

Dojo 的主页为 http://dojotoolkit.org/。Dojo 是一个 JavaScript 实现的开源 DHTML 工具包。Dojo 的最初目标是解决开发 DHTML 应用程序遇到的一些长期存在的历史问题。Dojo 先进的功能特点有：

（1）更容易地为 Web 页面添加动态能力，可以在其他支持 JavaScript 的环境中使用 Dojo；

（2）利用 Dojo 提供的组件，可以提升 Web 应用程序的可用性和交互能力；

（3）Dojo 设计的包加载机制和模块化的结构，能保持更好的扩展性，提高执行性能，减轻了用户开发的工作量，并保持一定的灵活性（用户可以自己编写扩展）；

（4）Dojo 在很大程度上屏蔽了浏览器之间的差异性，因此不必再担心 Web 页面是否在某些浏览器中可用；

（5）通过 Dojo 提供的工具，还可以为代码编写命令行式的单元测试代码；

（6）Dojo 的打包工具可以帮助优化 JavaScript 代码，并且只生成部署应用程序所需的最小 Dojo 包集合。

Dojo 主要由三大模块组成，分别是 Core、Dijit 和 DojoX。Core 提供 Ajax、事件、基于 CSS 的查询、动画以及 JSON 等相关操作 API。Dijit 是一个可更换皮肤，基于模板的 Web 界面控件库，包含许多简单易用的小部件（Widget）。DojoX 包括一些新颖的代码和控件，例如 DateGrid、Chart、离线应用和跨浏览器矢量绘图等。此外 Dojo 还包含一个工具库（Util）模块，该模块包含一个单元测试框架（Dojo Objective Harness，简写为 DOH）、从 Dojo 源代码中生成文档工具，以及 JavaScript 资源打包与压缩工具（Rhino）。这几个模块之间的相互关系如图 8.1 所示。

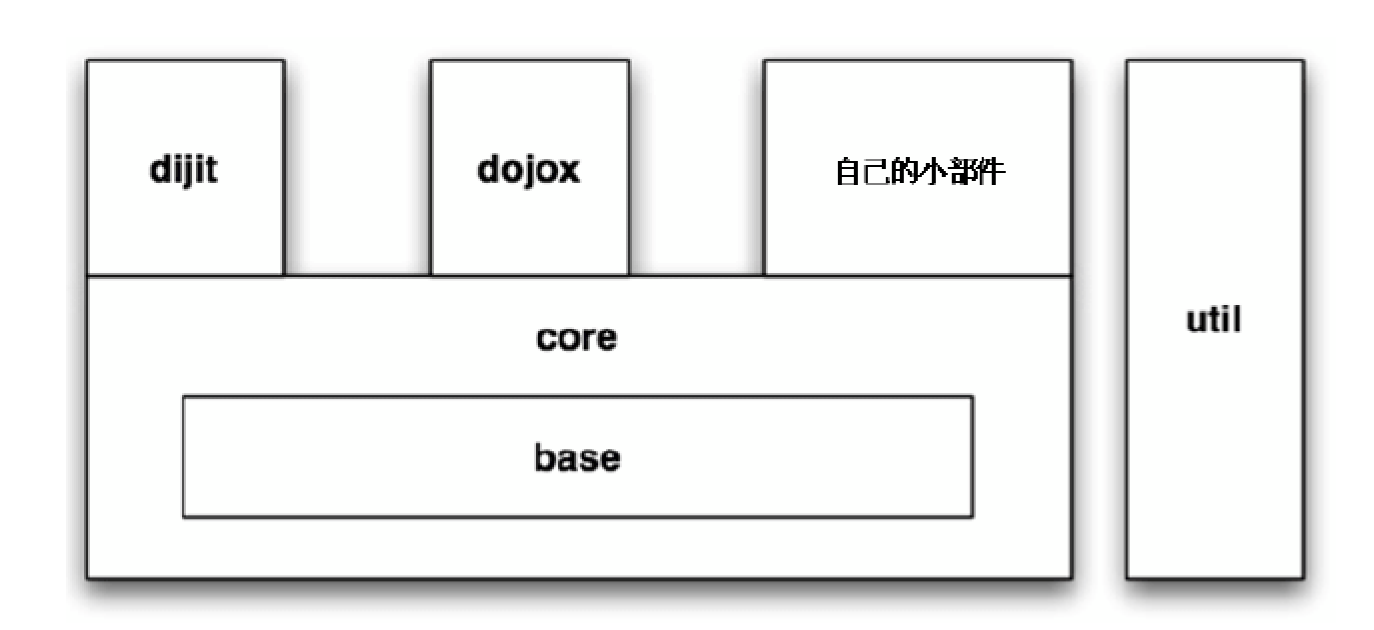


图 8.1　Dojo 的几个模块之间的关系

Dojo 的体系架构如图 8.2 所示，从总体上来看，Dojo 是一个分层的体系架构。最下面的一层是包系统，Dojo API 的结构与 Java 很类似，它把所有的 API 分成不同的包（package），当需要使用某个 API 时，只需要导入这个 API 所在的包。包系统上面一层是语言库，在这个语言库里包含一些语言工具 API，类似于 Java 的 util 包。再上一层是环境相关包，这个包的功能是处理跨浏览器的问题。



图 8.2　Dojo 的体系架构

Dojo 大部分代码都位于应用程序支持库（在图 8.2 中没有列出所有的包）。开发人员大部分时候都在调用这个层中的 API，比如用 IO 包可以进行 Ajax 调用。最上面的一层是 Dojo 的小部件系统，小部件指的是用户界面中的一个元素，比如按钮、进度条和树等。Dojo 的小部件基于 MVC 结构。它的视图作为一个 Template（模板）来进行存放，在 Template 中放置着 HTML 和 CSS 片段，而由控制器来对该 Template 中的元素进行操作。小部件不仅支持自定义的样式表，并且能够对内部元素的事件进行处理。用户在页面中只需要加入简单的标签

就可以使用。在这一层中存在数百个功能强大的小部件以方便用户使用，包括表格、树和菜单等。

Dojo 提供了上百个包，这些包分别放在 Dojo。Dijit 和 DojoX 这 3 个一级命名空间。由于 Dojo 包种类繁多，表 8.1 只列举了最常用的一些包及其功能，以方便读者有初步了解或供以后查阅。

表 8-1　Dojo 常用包

包名	功能
dojo/io	不同的输入输出传输方式。包括 Script、IFrame 等
dojo/dnd	拖放功能的辅助 API
dojo/string	这个包可以对字符串进行修整、转换为大写、编码、填充（pad）等处理
dojo/date	解析日期格式的有效助手
dojo/event	事件驱动 API，支持 AOP 开发，以及主题、队列的功能
dojo/back	用来撤销用户操作的栈管理器
dojo/rpc	与后端服务（例如理解 JSON 语法的 Web 服务）进行通信
dojo/colors	颜色工具包
dojo/data	Dojo 的统一数据访问接口，可以方便地读取 XML、JSON 等不同格式的数据文件
dojo/fx	基本动画效果库
dojo/regexp	正则表达式处理函数库
dijit/forms	表单控件相关的小部件库
dijit/layout	页面布局小部件库
dijit/popup	这个包用于以弹出窗口方式使用小部件
dojox/charting	用于在页面上画各种统计图表的工具包
dojox/collections	很有用的集合数据结构（List、Query、Set、Stack、Dictionary 等）
dojox/encoding	实现加密功能的 API（Blowfish、MD5、Rijndael、SHA 等）
dojox/math	数学函数（曲线、点、矩阵）
dojo/reflect	提供反射功能的函数库
dojox/storage	将数据保存在本地存储中（例如，在浏览器中利用 Flash 的本地存储来实现）
dojox/xml	XML 解析工具包

ArcGIS API for JavaScript 就构建于 Dojo 之上，本书后面的部分实践中也将使用 Dojo 来结合 OpenLayers 开发，因此在这里稍微详细介绍一下。

8.2　OpenLayers 的控件

虽然 OpenLayers 并不提供拖拉控件的界面来构建 Web 应用程序，但是却提供了许多控件，只需要一两行代码便可将这些控件加入到地图中。有些控件，例如地图平移控件以及前

面内容介绍的要素选择控件，并没有一个可视的实体，而另外一些控件，例如鹰眼地图、图层切换器等，具有可视实体。同时，控件框架是可扩展的，可使用其他开发者编写的控件，当然也可以编写自己的控件。

在浏览器中查看“地图控件实例”（http://dev.openlayers.org/releases/OpenLayers-2.13.1/examples/ controls.html），如图 8.3 所示。通过该实例可了解其中一些控件的使用。



图 8.3　地图控件实例

当创建一个 OpenLayers 地图对象时，就已经包含了地图基本操作控件——OpenLayers. Control.Navigation。如果希望改变其中的控件，在当初始化地图对象时可传入希望加入的控件列表。对于来加入图 8.3 所显示的控件，可通过如下代码来实现。

```
map = new OpenLayers.Map('map', {
    controls: [
        new OpenLayers.Control.Navigation(),
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.LayerSwitcher({'ascending':false}),
        new OpenLayers.Control.Permalink(),
        new OpenLayers.Control.ScaleLine(),
        new OpenLayers.Control.Permalink('permalink'),
        new OpenLayers.Control.MousePosition(),
        new OpenLayers.Control.OverviewMap(),
        new OpenLayers.Control.KeyboardDefaults()
    ],
    numZoomLevels: 6
});
```

我们可以以上面的代码为基础，根据实际再增减控件。注意，在上面没有列出的一个名

为 OpenLayers.Control.Zoom 的控件，在地图上的显示如图 8.4 所示，只是一个简单的“+”与“-”按钮，比较适合移动应用。



图 8.4　OpenLayers.Control.Zoom 控件

可以通过修改样式表来自定义控件。例如可使用如下样式代码改变鹰眼地图控件的外观。

```
<style>
    .olControlOverviewMap {
        background:#ff0000;
        width:200px;
    }

    .olControlOverviewMapElement {
        background:#00ff00 !important;
    }
</style>
```

8.3　基于属性值符号化图层

基于要素的属性值而符号化的地图称为专题图。专题图是用于分析和表现数据的一种强有力的方式。用户可以通过使用专题图的方式将数据图形化，使数据以更直观的形式在地图上体现出来。当使用专题渲染在地图上显示数据时，可以清楚地看出这数据记录上难以发现的模式和趋势，为用户的决策提供依据。专题图所表示的内容往往是普通地图上没有的而在地面上看不到或无法直接量算的要素和现象，例如气候、人口分布、各种统计图等。

制作专题图是根据某个特定专题对要素图层“渲染”的过程。所谓的专题渲染，就是以某种图案或颜色填充来表明要素对象的某些信息，也就是说，这类渲染存在着主题，经过这样渲染的地图就是专题图。利用 OpenLayers 可根据属性数据中特定的值来赋给要素对象颜色、图案或符号，从而创建不同的专题图。该专题可以是一个或多个专题变量。所谓专题变量就是指在地图上显示的数据。一个专题变量可以是一个字段或表达式。

当然，使用 OpenLayers 代码根据要素属性值符号化图层，并不是专题图制作的唯一途径。正如前几章介绍的，也可以在服务器端利用 SLD 或 CartoCSS 制作专题图。服务器端地图更

适合于高级制图效果。而且当图层中包含成千上万个要素时，在服务器端制图具有更好的性能。

既然存在这么多服务器端制图工具，那么又为什么使用 OpenLayers 在客户端制图呢？主要原因是在客户端制图具有更大的灵活性与交互性。

（1）灵活性来自于数据与样式的分离。对于同一个 Web 服务，在客户端可以利用同一数据源以制作多种不同样式的专题图，而不需要在服务器端发布多个 Web 服务。

（2）交互性来自可以通过客户端代码动态重新分类与符号化图层。例如，可以允许用户从一个相等的间隔分类切换到分位数的分类，或调节比例符号的最大尺寸，所有这些都不需要与服务器交互或增设新的 Web 服务。

8.3.1　在 OpenLayers 中读取属性值

要基于属性值来设置样式规则，那么第一步自然就是获取要素的属性值。那么在 OpenLayers 中如何获取要素的属性值呢？

假设有一个城市的地铁线要素图层，该图层中各字段如图 8.5 所示。现在有一个需求是用不同的颜色分别绘制不同的地铁线。这里使用最简单的方式，图层中的 COLOR 字段包含了颜色值。

	ID	LINEASUB	COLOR
68	69	LINEA A	#46C7FA
69	70	LINEA A	#46C7FA
37	34	LINEA B	#E81D1A
38	35	LINEA B	#E81D1A
39	36	LINEA B	#E81D1A

图 8.5　一城市地铁线要素数据

在 OpenLayers 中，可使用“${<fieldName>}”获取来自名为“fieldName”字段的属性值。因此可使用如下代码，实现用每条地铁线要素“COLOR”字段的值绘制该要素。

```
var geojson_layer = new OpenLayers.Layer.Vector("GeoJSON", {
    projection: new OpenLayers.Projection("EPSG:900913"),
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "red_de_subte.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: new OpenLayers.StyleMap({
        'strokeWidth': 4,
        'strokeColor': '${COLOR}'
    })
});
```

上述代码从 red_de_subte.geojson 文件创建了一个矢量图层。为样式化该图层，代码从 COLOR 字段读取十六进制的值，然后将其赋予样式映射对象的 strokeColor 属性。

8.3.2 独立值专题图

独立值专题图是一种比较简单的专题图。它使用不同的颜色、符号或线型来显示不同的数据。根据独立值绘制的专题图有助于强调数据的类型差异而不是显示定量信息（例如给定区域内的商店类型、分区类型等）。因此，当用户只需要使用单一的数据值来渲染时，可使用独立值专题图。

例如对于上面的地铁线要素数据，假设没有 COLOR 字段，那么则需要根据每条线的名称提供一个唯一的颜色值。实现代码如下：

```
var subteStyleMap = new OpenLayers.StyleMap({
    'strokeWidth': 4
});

var lookup = {
    "LINEA A": {strokeColor: "#46C7FA"},
     "LINEA B": {strokeColor: "#E81D1A"},
     "LINEA C": {strokeColor: "#4161BA"},
     "LINEA D": {strokeColor: "#599C65"},
     "LINEA E": {strokeColor: "#65018A"},
     "LINEA H": {strokeColor: "#FAF739"}
}

subteStyleMap.addUniqueValueRules("default", "LINEASUB", lookup);

var geojson_layer = new OpenLayers.Layer.Vector("GeoJSON", {
     projection: new OpenLayers.Projection("EPSG:900913"),
     strategies: [new OpenLayers.Strategy.Fixed()],
     protocol: new OpenLayers.Protocol.HTTP({
          url: "red_de_subte.geojson",
          format: new OpenLayers.Format.GeoJSON()
     }),
     styleMap: subteStyleMap
});
```

在上述代码中，首先创建了一个基本的样式映射对象，该对象只指定了画笔宽度。然后创建了一个名为 lookup 的非常简单的 JavaScript 对象，用于在地铁线与期望的颜色间建立映射关系。最重要的是利用样式映射对象的 addUniqueValueRules 方法，将该颜色映射与图层中的 LINEASUB 字段连接起来。

8.3.3　等级符号专题图

等级符号专题图使用不同大小的符号来表示不同的值，而且符号大小与数据值成比例。等级符号专题图对于阐明定量信息（如由高到低依次变化）很有用处。符号的大小与该要素对应的数值成比例，数值越大符号就越大，数值越小符号就越小。因此，等级符号专题图最适合数据值数据。

图 8.6 所示为南美洲最大地铁系统图层的属性。

	X	Y	CITY	COUNTRY	YEAR	LENGTHKM	STATIONS	PASSDAY	LINK	PHOTO
0	-46.655	-23.549	Sao Paulo	Brazil	1974	74	67	2400000	http://en.wikip...	<iframe src="h...
1	-70.687	-33.439	Santiago	Chile	1975	102	108	1750000	http://en.wikip...	<iframe src="h...
2	-66.917	10.5	Caracas	Venezuela	1983	60	50	1330000	http://en.wikip...	<iframe src="h...
3	-43.1964	-22.908	Rio de Janeiro	Brazil	1979	42	34	581000	http://en.wikip...	<iframe src="h...
4	-58.382	-34.603	Buenos Aires	Argentina	1913	51	81	844000	http://en.wikip...	<iframe src="h...
5	-75.575	6.236	Medellin	Colombia	1995	28	26	466000	http://en.wikip...	<iframe src="h...

图 8.6　南美洲最大地铁系统图层属性

如果不做特别的处理，那么所有站点大都用一样大小的符号表示，如图 8.7 中的左图所示。假设需要按比例大小的符号，这样某地铁系统的站点（STATIONS 字段）越多，则使用较大的地图符号，如图 8.7 中的右图所示。

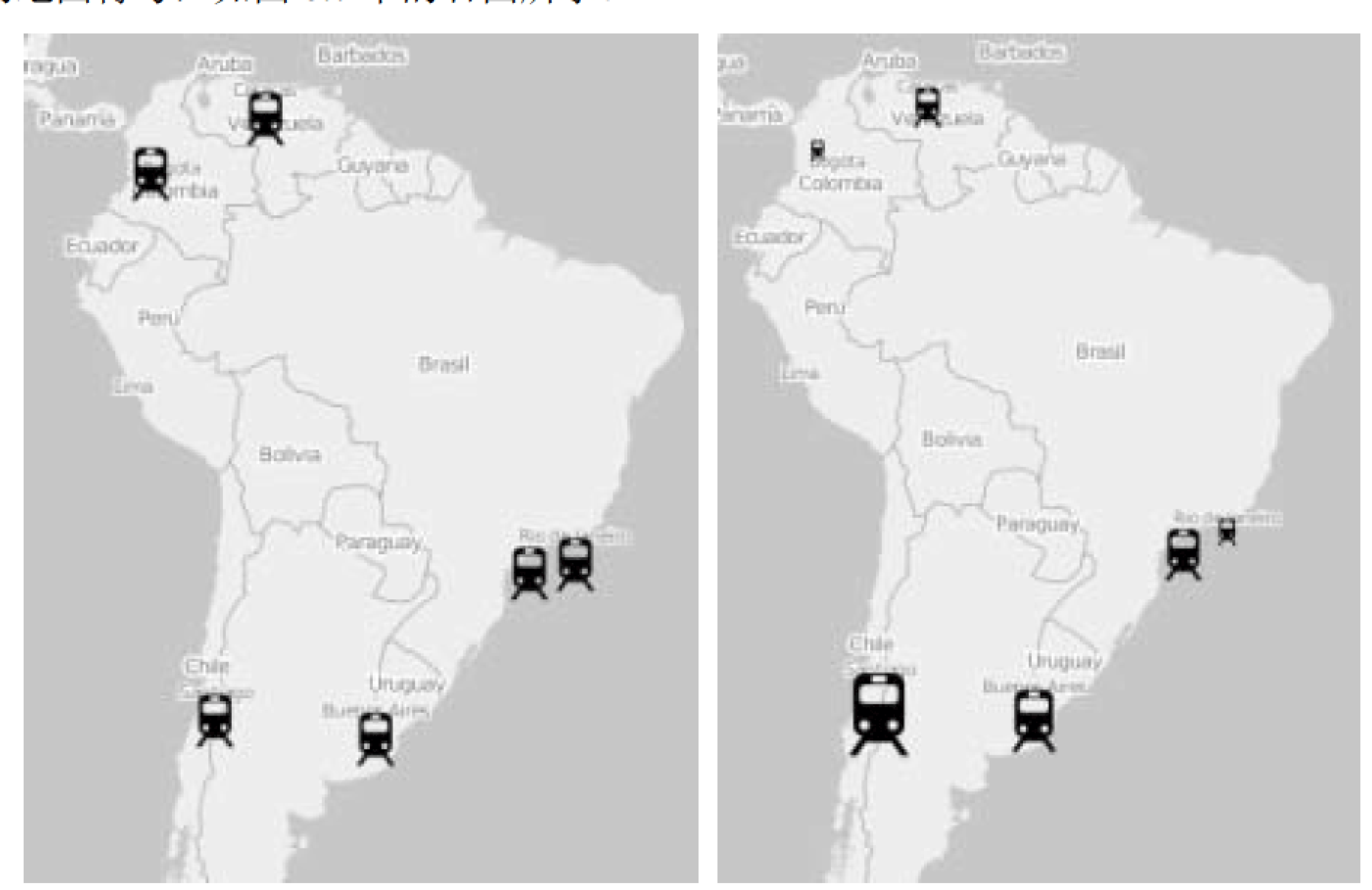


图 8.7　使用统一符号与等级符号分别制图的效果

要在 OpenLayers 中实现这种等级符号，需要定义一些数据函数，根据每个要素的 STATIONS 属性值来设置符号的大小。下面的代码将某要素的 STATIONS 值除以 80，然后乘以 30，得到该要素使用符号以像素为单位的高与宽值。这就意味着，如果某地铁系统有 80 个站点，那么将使用 30 像素宽的符号，如果某地铁系统的站点少于 80 个，那么符号的宽度

也将小于 30 个像素，多余 80 个站点的地铁系统的符号将大于 30 个像素（当然，数字 80 和 30 完全是任意的，可以根据实际情况进行适当调整）。

```
var context = {
    getSize: function(feature) {
        return feature.attributes.STATIONS/80 * 30;
    }
};

var template = {
    externalGraphic: 'metro.svg',
    graphicWidth: '${getSize}',   // 使用 context.getSize(feature)
    graphicHeight: '${getSize}',   // 使用 context.getSize(feature)
    graphicYOffset: 0
};

var metroStyle = new OpenLayers.Style(template, {context: context});
var metroStyleMap = new OpenLayers.StyleMap({'default': metroStyle});

// 定义地铁系统 GeoJSON 图层
var metroLayer = new OpenLayers.Layer.Vector("Metro lines", {
    projection: toProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "metro.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: metroStyleMap
});
```

在上面定义 OpenLayers 样式时，使用了两个新的对象，分别是模板与上下文。上下文对象包含了 getSize 函数，用于从当前要素中读取 STATIONS 属性值。当 graphicWidth 与 graphicHeight 定义时，调用了该函数。

8.3.4 范围专题图

范围专题图是按照设置的范围对数据进行分类与显示。这些分类用颜色和图案进行渲染。范围专题图能够通过点、线和区域来说明数值，在反映数值和地理区域的关系（如销售数字、家庭收入），或显示比率信息如人口密度（人口除以面积）时是很有用的。

对于如何确定范围的边界值，也就是如何分类，可使用相等间隔、分位数、自然间断点分级法、几何间隔或其他任意方式。

例如对于上面的南美洲地铁系统，可以将包含 100 个站点以上的要素使用深红色图标表示 50~100 个站点之间的要素使用红色图标，而对于少于 50 个站点的要素，使用粉红色图标，如图 8.8 所示。



图 8.8　范围专题图

要在 OpenLayers 中进行范围专题制图，需要为每类定义一个 OpenLayers.Rule 对象，即规则对象。在该规则对象中，定义该类数值的上限与下限。由于 OpenLayers 的规则系统还可以用于其他多种查询，因此代码相对比较复杂。下面的代码将南美洲最大地铁系统图层分成三类，每类使用不同颜色的 SVG 符号。

```
var metroStyle = new OpenLayers.Style(
    // 第一个参数是一基本的符号
    // 所有其他符号都扩展该符号
    {
        graphicWidth: 25,
        graphicHeight: 25,
        graphicYOffset: 0,
    },
    // 第二个参数是一组规则
    {
        rules: [
            new OpenLayers.Rule(
            {
                filter: new OpenLayers.Filter.Comparison({
                type: OpenLayers.Filter.Comparison.GREATER_THAN_OR_EQUAL_TO,
```

```
                    property: "STATIONS",
                    value: 100 }),
                    symbolizer: {"Point": {externalGraphic: 'metro_high.svg'}}
                }),

                new OpenLayers.Rule({filter: new OpenLayers.Filter.Logical({
                    type: OpenLayers.Filter.Logical.AND,
                    filters: [
                        new OpenLayers.Filter.Comparison({
                            type: OpenLayers.Filter.Comparison.LESS_THAN,
                            property: "STATIONS",
                            value: 100
                        }),
                        new OpenLayers.Filter.Comparison({
                            type: OpenLayers.Filter.Comparison.GREATER_THAN_OR_ EQUAL_TO,
                            property: "STATIONS",
                            value: 50
                        })
                    ]
                    }),
                    symbolizer: {"Point": {externalGraphic: 'metro medium.svg'}}
                }),

                new OpenLayers.Rule({filter: new OpenLayers.Filter.Comparison({
                    type: OpenLayers.Filter.Comparison.LESS THAN,
                    property: "STATIONS",
                    value: 50
                    }),
                    symbolizer: {"Point": {externalGraphic: 'metro low.svg'}}
                }),
                new OpenLayers.Rule({
                    // 如果没有利用其他规则，则使用本规则
                    elseFilter: true,
                    symbolizer: {
                        externalGraphic: "metro.svg"
                    }
                })
            ]
        }
);

var metroStyleMap = new OpenLayers.StyleMap({'default': metroStyle});
```

```
// 定义地铁系统 GeoJSON 图层
var metroLayer = new OpenLayers.Layer.Vector("Metro lines", {
    projection: toProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: "metro.geojson",
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: metroStyleMap
});
```

虽然上述代码看起来很多，但是逻辑还是非常清楚，就是定义规则并将规则加入到 OpenLayers 的样式中。对于中间分类（50~100 个站点）规则的代码相对最复杂，因为需要同时定义上限与下限。要实现该功能，需要将两个比较过滤条件进行结合，形成一个逻辑过滤条件。

上述代码是通过“手动”来定义范围与类，也就是手动添加分类间隔并设置适合数据的类范围。如果要使用相等间隔、分位数、自然间断点分级法与几何间隔等，那么在定义规则之前，需要计算以确认分隔值。可以手工计算，也可以通过 JavaScript 代码动态计算。

8.3.5　根据属性限制要素的显示

有时候希望根据一些属性值或组合值，仅仅显示数据集中部分要素。在 ESRI 的 ArcMap 中，这称为定义查询。例如以下情况：

（1）只希望显示人口数超过某个阈值的城市。

（2）许多数据集（例如道路和街道数据集）包含要素（类）的子集，可能希望独立于其他要素而为各个道路类定义地图图层。

（3）在另一种情况下，可能具有大型企业级的数据库，在该数据集中包含覆盖较大区域（例如全国或整个州）的成千上万个要素。不过在地图中，可能只希望使用这些数据的一个子集。

在前面学习的 OpenLayers.Rule 可用于实现该功能。

假设，对于南美洲地铁系统图层，现需要只显示 COUNTRY 属性值为“Brazil”的要素，可使用如下代码来实现。

```
var metroStyle = new OpenLayers.Style(
    // 第一个参数是一基本符号，其他符号在此基础上进行扩展
    {
        graphicWidth: 25,
        graphicHeight: 25,
        graphicYOffset: 0,
```

```
        },
        // 第二个参数是一组规则
        {
            rules: [
                new OpenLayers.Rule({
                    filter: new OpenLayers.Filter.Comparison({
                        type: OpenLayers.Filter.Comparison.EQUAL_TO,
                        property: "COUNTRY",
                        value: "Brazil"
                    }),
                    symbolizer: {"Point": {externalGraphic: 'metro.svg'}}
                })
            ]
      }
);

var metroStyleMap = new OpenLayers.StyleMap({'default': metroStyle});
```

上面的粗体代码行用于查找 COUNTRY 属性值为“Brazil”的要素。要注意的是，在上面的代码中，仅仅根据属性值进行了过滤，并没有进行空间查询，也就是说没有用巴西的边界图形去获得位于其中的要素。不过，OpenLayers 也提供了空间查询过滤。

此外，也可以根据数值型的属性进行过滤。例如假设只希望显示站点数在 75 个以上的地铁系统要素，可使用如下代码来实现。

```
var metroStyle = new OpenLayers.Style(
        {
            graphicWidth: 25,
            graphicHeight: 25,
            graphicYOffset: 0,
        },
        {
            rules: [
                new OpenLayers.Rule({
                    filter: new OpenLayers.Filter.Comparison({
                    type: OpenLayers.Filter.Comparison.GREATER_THAN_OR_EQUAL_TO,
                    property: "STATIONS",
                    value: 75 }),
                    symbolizer: {"Point": {externalGraphic: 'metro.svg'}}
                })
            ]
        }
);
```

```
var metroStyleMap = new OpenLayers.StyleMap({'default': metroStyle});
```

8.4　实践 13：使用 OpenLayers 与 Dojo 进行专题制图

在本实践中，将综合利用本章介绍的专题制图以及 OpenLayers 的控件，展现南美大型地铁系统，并利用 Dojo 这一强大的 JavaScript 框架，进行页面布局。本实践使用的所有资源位于下载文件的“Codes\Walkthrough13”文件夹中。

8.4.1　页面布局

Web 应用的页面布局一直是令 Web 开发者头疼的一件事情。在 Web 2.0 的时代，Web 页面布局的设计越来越多样化，仅仅依靠表格和 CSS 控制的页面布局已难满足用户的需求。为此 Dojo 提供了一系列的布局小部件辅助 Web 开发人员实现复杂的页面布局。

Dojo 中提供的布局小部件可以分为如下 3 类，主要是在 Dijit 中。

（1）面板：盛放和显示大块的内容，包括文本、图片、图表以及其他小部件。这类的布局小部件有 ContentPane、FloatingPane 与 ExpandoPane 等，后两者在 DojoX 中；

（2）对齐方式容器：用以盛放屏面类小部件，并且可以设置这些小部件的排列方式。这类的布局小部件有 BorderContainer、LayoutContainer 与 SplitContainer 。其中 BorderContainer 是在 Dojo1.1 中引进的轻量级组件，有取代 LayoutContainer 与 SplitContainer 小部件之势。目前对于 Dojo 不推荐使用 LayoutContainer 与 SplitContainer widgets；

（3）堆叠容器：此类的小部件可以把前小部件层叠在一起，而一次只显示一个屏面。这类的布局小部件有 AccordionContainer、TabContainer 与 StackContainer 等。

在设计页面布局时，首先应选择页面整体的框架：上下两栏、左右两栏、上中下三栏、左中右三栏、上一栏下两栏或上下左右中五栏等。在以往的 DIV + CSS 设计布局时，虽然可以轻松地将前 5 种布局实现，但如果要实现最后一种五栏的布局却有些困难。并且设计后的布局间的比例或者每栏的大小都是固定的，当一栏的内容超出栏宽或高时，只能通过左右拉条或者下拉条的拖动来显示超出的内容。可以说既麻烦又不美观。

新建一个名为 Walkthrough13 的文件夹，在其中加入名为 ThematicMap.html 的文件。该文件的内容如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
user-scalable=0">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>实践 13：专题制图</title>
```

```
        <link rel="stylesheet" href="http://demos.dojotoolkit.org/dijit/themes/tundra/tundra.css">
        <style>
            html, body, #main {
                height: 100%;
                width: 100%;
                padding: 0;
                margin: 0;
            }
            #map {
                border: 1px solid black;
            }
        </style>
        <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js"></script>
        <script src="http://demos.dojotoolkit.org/dojo/dojo.js" data-dojo-config="async: true"></script>
        <script src="src.js"></script>
    </head>
    <body class="tundra">
        <div id="main" data-dojo-type="dijit/layout/BorderContainer" data-dojo-props="design:'headline',
                gutters:false">
            <div data-dojo-type="dijit/layout/ContentPane" data-dojo-props="region:'top'"
                style="height: 10%;">
                <h3>南美大型地铁系统</h3>
            </div>

            <div id="map" data-dojo-type="dijit/layout/ContentPane" data-dojo-props="region:'center'">
            </div>

            <div data-dojo-type="dijit/layout/ContentPane" data-dojo-props="region:'right', splitter:'true'"
                                style="width: 50%;">
                <div data-dojo-type="dijit/layout/BorderContainer" data-dojo-props="design:'headline',gutters:
                                false">
                    <div id="summaryLabel" data-dojo-type="dijit/layout/ContentPane" data-dojo-props=
                            "region:'top', splitter:'true'" style="height: 30%;">
                        <p>在地图上单击地铁系统，获取更多信息。</p>
                    </div>
                    <div id="metroImage" data-dojo-type="dijit/layout/ContentPane" data-dojo-props=
                            "region:'center'">
                    </div>
                </div>
            </div>

            <div data-dojo-type="dijit/layout/ContentPane" data-dojo-props="region:'bottom'" style="height: 32px;">
```

```
                <img src="metro  legend.png"/>
            </div>
        </div>
    </body>
    </html>
```

要使用 Dojo，可以从其官方网站 http://dojotoolkit.org/download/下载源代码，也可以使用 CDN 服务器部署的内容，主要 CDN 一个是谷歌，地址如下：

http://ajax.googleapis.com/ajax/libs/ dojo/1.10.3/dojo/dojo.js

在我国有百度，地址是 http://libs.baidu.com/dojo/1.8.0/dojo.js。此外，还可以直接使用 Dojo 官网本书提供的在线代码。上述代码使用的方式是后者。

要使用 Dojo，首先需要增加引用其提供的样式文件，代码如下：

```
<link rel="stylesheet" href="http://demos.dojotoolkit.org/dijit/themes/tundra/tundra.css">
```

Dojo 提供了四组样式，分别是 claro、tundra、soria 以及 nihilo，每种是一组定义用户界面的字体、颜色与大小等设置。读者可以通过如下地址来了解每组样式中不同 Dojo 小部件的显示情况：

```
http://archive.dojotoolkit.org/nightly/dojotoolkit/dijit/themes/themeTester.html
```

第二步当然是要引用 Dojo 的 JavaScript 代码文件，代码如下：

```
<script src="http://demos.dojotoolkit.org/dojo/dojo.js" data-dojo-config="async: true"></script>
```

第三步就是利用 Dojo 的页面布局小部件进行页面设计。一般是面板小部件来组织“原子”小部件，例如按钮、文本框等，然后利用容器小部件将面板放置在页面的不同区域。

在 Dojo 中，最重要的容器容器小部件是 BorderContainer 容器。每个 BorderContainer 实例可以至多允许包含 5 个不同区域：上、下、左、右与中，当然也可以只包含其中几个区域。如果某个区域需要进一步分解，则可将 BorderContainer 中的一个区域设置为另一 BorderContainer 实例。

每个 BorderContainer 都有两种不同的方式安排其子元素的位置，可以通过其“design”属性来控制。该属性的值可以是 headline 或 sidebar。当该属性设置为 headline 时，上、下两个区域优先设置，并且它们的宽度就会与整个 BorderContainer 的宽度相同，称为“上中下结构”；如果设置为 sidebar，那么左、右两区域优先设置，并且它们的高度与整个 BorderContainer 的高度一致，称为“左中右结构”，如图 8.9 所示。

图 8.9　上中下结构与左中右结构的页面布局

BorderContainer 容器中的每个区域一般对应一个面板（ContentPane），也可以是一容器。

在本实践中，使用的是上中下结构，不过没有左部分，并且在右边部分也是一 BorderContainer 容器实例，而在该容器实例中只有上中部分。

从上述代码中可以看到样式的定义中设置了 width 和 height，这里也是需要特别注意的地方，BorderContainer 节点需要设置 width 和 height（这里是通过 head 中设置样式代码段中的 #main 来设置的）。同时左右两个子节点可以设置宽度，而上下两个子节点可以设置高度。对于中间区域的子节点不需要设置大小，减去 4 个边界区域占有的空间，剩下的就是中间区域。

如果用户不喜欢页面布局的尺寸怎么办？可以用可拖曳隔条将修改尺寸的权利交给他们。拖曳隔条是一个可以手动拖曳改变两侧区域尺寸的边框，可以通过设定 ContentPane 的 splitter="true"来实现。这让依赖于四周区域的尺寸来决定自己长宽的中心区域也可拖曳了。如果想让用户的拖曳有个限度，可以通过指定 minSize 或 maxSize 的属性值来实现。尺寸限制了指定区域的长宽。其默认最小值和最大值分别是 0 和 Infinity，表示没有限制。

BorderContainer 的 liveSizing 属性设定了面板在拖曳过程中是否需要重绘。设为 true 时可以帮助用户即时地看到拖曳的效果。但如果 ContentPane 里有大量 HTML 的话，整个页面会变得非常慢。此外，BorderContainer 可以在浏览器的 cookie 中保存拖曳后的位置。只需要将 persist 属性设定为 true 就可以实现，这样用户不必每次都更改边框的长宽。

8.4.2 代码设计

为了保证页面小而整洁，本实践将 JavaScript 代码放置在 src.js 文件中。

（1）创建代码框架。

在 src.js 文件中，首先加入如下框架代码：

```
    require(["dojo/parser", "dijit/layout/BorderContainer", "dijit/layout/ContentPane", "dojo/domReady"], function
(parser) {
        parser.parse();
    });
```

在上面的代码中，调用了 Dojo 的全局函数 require。require 函数需要两个参数，第一个参数是依赖项，第二个参数是一回调函数。

require 函数的第一个参数指定的是执行代码所依赖的，包括两类，一类是真正的依赖的类，另一类是插件，比如 dojo/dom、dojo/fx、dojo/domReady!等。对于依赖的类，如果不存在，Dojo 就会根据目录结构去加载。当加载完成之后，将执行回调函数。插件是用来扩展加载器功能的。插件的加载方式和常规模块没什么区别，只是在模块标识符的结尾使用了特殊符号!来表明它的请求时插件请求。Dojo 默认带有一些插件，其中有 4 个最重要的插件是：dojo/text、dojo/i18n、dojo/has 与 dojo/domReady。对于 dojo/domReady 插件，意思是当 DOM 解析完毕以后再执行回调函数，这样就可以确保在执行任何代码前 DOM 可用。

在回调函数中的参数依次是 require 函数的第一个参数指定的依赖类的别名，当然可以指

定为不重复的变量名即可，但是为了代码的可读性、可维护性以及一致性，最好是对于同一个模块使用统一的别名。

在回调函数中，调用 dojo/parser 功能模块的 parse 函数解析小部件标签属性。由于页面中通过 data-dojo-type 标签属性使用了 Dojo 的小部件，但是这并不是标准的 HTML，浏览器不能直接对其进行解析。因此需要在页面加载完成以后，对整个页面的所有标签属性进行解析，将其转换为浏览器可以识别的标记。因此需要调用 dojo/parser 功能模块的 parse 函数。

上述代码框架首先确保 DOM 加载完成后，执行上述代码，并将页面中用到的 Dojo 的小部件解析为浏览器能直接使用的标记。其他代码都加入到 parser.parse();代码行的后面。

（2）创建地图与控件。

在 parser.parse();代码行后面，加入如下代码：

```
var fromProjection = new OpenLayers.Projection('EPSG:4326'); // WGS84 坐标系
var toProjection = new OpenLayers.Projection('EPSG:900913'); // Web 墨卡托投影

// 创建地图与控件
var map = new OpenLayers.Map('map', {
      projection: toProjection,
      controls: [
         new OpenLayers.Control.Navigation(),
         new OpenLayers.Control.LayerSwitcher({ 'ascending': false }),
         new OpenLayers.Control.Attribution()
      ]
});
```

在上述代码中，加入了一个图层切换器控件，通过该控件用户可以控制图层的关闭与显示。同时也加入了一个属性控件，用于显示版权文字。

（3）加入基础底图。

创建地图与控件之后，加入如下代码，在地图中加入 3 个可选的切片地图。

```
// 增加可选基础底图，每次只能显示一个底图。
// 来自 Mapnik 的 OpenStreetMap
map.addLayer(new OpenLayers.Layer.OSM('Mapnik'));

// MapQuest 切片
map.addLayer(new OpenLayers.Layer.OSM('MapQuest Open',
      ['http://otile1.mqcdn.com/tiles/1.0.0/osm/${z}/${x}/${y}.png',
      'http://otile2.mqcdn.com/tiles/1.0.0/osm/${z}/${x}/${y}.png',
      'http://otile3.mqcdn.com/tiles/1.0.0/osm/${z}/${x}/${y}.png',
      'http://otile4.mqcdn.com/tiles/1.0.0/osm/${z}/${x}/${y}.png'],
      { attribution: "© <a href='http://www.openstreetmap.org/'>OpenStreetMap</a> and contributors, under
an <a href='http://www.openstreetmap.org/copyright' title='ODbL'>open license</a>. Tiles Courtesy of <a
```

```
href='http://www.mapquest.com/'>MapQuest</a> <img src='http://developer.mapquest.com/content/osm/mq logo.
png'>" }));

    // 来自 Stamen 工作室的切片
    map.addLayer(new OpenLayers.Layer.OSM('Stamen toner',
        ['http://tile.stamen.com/toner/${z}/${x}/${y}.png'],
        {
            attribution: "© <a href='http://www.openstreetmap.org/'>OpenStreetMap</a> and contributors, under
an <a href='http://www.openstreetmap.org/copyright' title='ODbL'>open license</a>. Toner style by <a href=
'http://stamen.com'>Stamen Design</a>",
            "tileOptions": { "crossOriginKeyword": null }
    }));
```

虽然同时都将 3 个切片地图作为图层加入到了地图中，但是 OpenLayers 知道它们是基础底图，因此每次只显示一个。图层切换器控件会检测到这些信息，并允许用户在这三者之间进行切换，如图 8.10 所示。

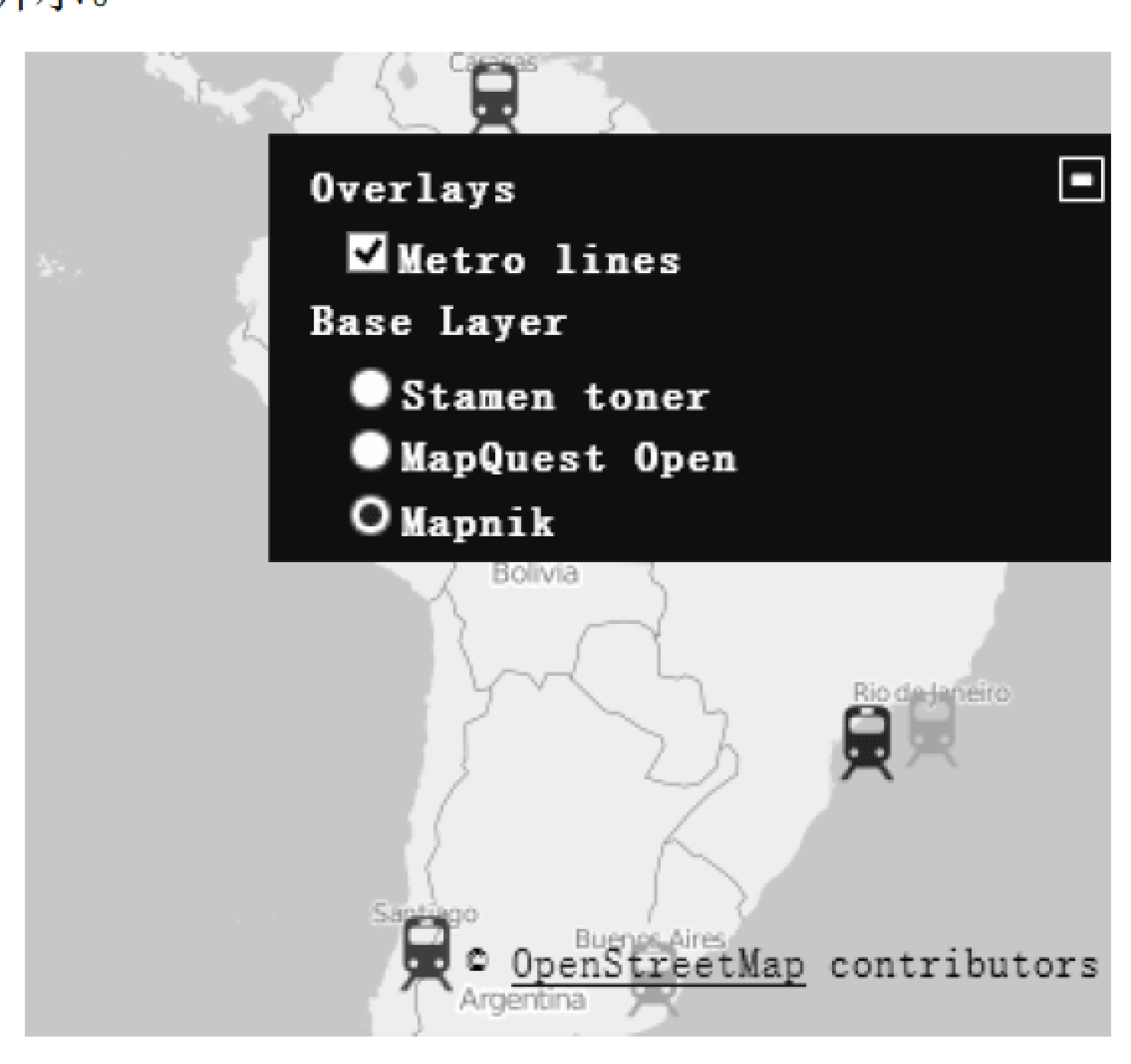


图 8.10　图层切换器控件

（4）设置专题符号。

加入如下代码，用于地铁系统图层专题制图。

```
    // 设置范围专题图分类与样式规则
    var metroStyle = new OpenLayers.Style({
        graphicWidth: 25,
        graphicHeight: 25,
        graphicYOffset: 0,
    },
    {
```

```
rules: [
  // 对于每天超过 200 万乘客要素的规则
  new OpenLayers.Rule({
      filter: new OpenLayers.Filter.Comparison({
          type: OpenLayers.Filter.Comparison.GREATER THAN OR EQUAL TO,
          property: 'PASSDAY',
          value: 2000000
      }),
      symbolizer: { 'Point': { externalGraphic: 'metro high.svg' } }
  }),

  // 对于每天乘客数量在 100~200 万之间要素的规则
  new OpenLayers.Rule({
      filter: new OpenLayers.Filter.Logical({
          type: OpenLayers.Filter.Logical.AND,
          filters: [
            new OpenLayers.Filter.Comparison({
                type: OpenLayers.Filter.Comparison.LESS_THAN,
                property: 'PASSDAY',
                value: 2000000
            }),
            new OpenLayers.Filter.Comparison({
                type: OpenLayers.Filter.Comparison.GREATER_THAN_OR_EQUAL_TO,
                property: 'PASSDAY',
                value: 1000000
            })
          ]
      }),
      symbolizer: { 'Point': { externalGraphic: 'metro medium.svg' } }
  }),

  // 对于每天少于 100 万乘客要素的规则
  new OpenLayers.Rule({
      filter: new OpenLayers.Filter.Comparison({
          type: OpenLayers.Filter.Comparison.LESS_THAN,
          property: 'PASSDAY',
          value: 1000000
      }),
      symbolizer: { 'Point': { externalGraphic: 'metro_low.svg' } }
  }),

  // 默认规则
```

```
        new OpenLayers.Rule({
            elseFilter: true,
            symbolizer: {
                externalGraphic: 'metro.svg'
            }
        })]
});
```

（5）设置被选中要素使用的样式。

加入如下代码，设置被选中要素使用的样式。

```
// 对于被选中的要素使用黄色符号
var selectedMetroStyle = new OpenLayers.Style({
    externalGraphic: 'metro_selected.svg',
    graphicWidth: 25,
    graphicHeight: 25,
    graphicYOffset: 0
});

var metroStyleMap = new OpenLayers.StyleMap({ 'default': metroStyle, 'select': selectedMetroStyle });
```

上述最后一行代码将默认样式与被选中样式进行组合，一并加入到了样式映射对象中。

（6）矢量图层创建。

加入如下代码，创建矢量图层，并将其加入到地图中。

```
// 定义地铁系统 GeoJSON 图层
var metroLayer = new OpenLayers.Layer.Vector('Metro lines', {
    projection: toProjection,
    strategies: [new OpenLayers.Strategy.Fixed()],
    protocol: new OpenLayers.Protocol.HTTP({
        url: 'metro.geojson',
        format: new OpenLayers.Format.GeoJSON()
    }),
    styleMap: metroStyleMap
});

// 将地铁系统图层加入到地图中，并设置地图中心点
map.addLayer(metroLayer);
map.setCenter(new OpenLayers.LonLat(-60, -25).transform(fromProjection, toProjection), 3);
```

（7）加入选择控件。

要实现要素选择功能，首先需要加入选择控件，用于监听要素选择与取消选择事件。代码如下：

```
// 监听要素选择与取消选择事件
var selectControl = new OpenLayers.Control.SelectFeature([metroLayer], {
    onSelect: onFeatureSelect,
    onUnselect: onFeatureUnselect
});

map.addControl(selectControl);
selectControl.activate();
```

如果希望当用户单击某要素时，改变符号或执行其他操作，可使用选择控件。这样便可以不需要编写代码去监听点击事件或切换符号了，这些工作都由选择控件来完成。需要编写的代码是定义两个函数，分别用于当选择与取消选择时，还需要的进一步功能。上面的代码指示，当要素被选择时，调用 onFeatureSelect 函数；取消选择时，将调用 onFeatureUnselect 函数。

（8）处理要素选择事件。

加入如下代码，用于处理要素选择事件。

```
// 处理要素选择事件，获取要素属性，构建 HTML
function onFeatureSelect(feature) {
    // 获取要素属性，对于没有的属性使用默认值
    var featureName = feature.attributes.CITY || '无名要素';
    var country = feature.attributes.COUNTRY || '(不明确)';
    var year = feature.attributes.YEAR || '(不明确)';
    var passengers = feature.attributes.PASSDAY || '(不明确)';
    var stations = feature.attributes.STATIONS || '(不明确)';
    var length = feature.attributes.LENGTHKM || '(不明确)';
    var link = feature.attributes.LINK || 'http://www.wikipedia.org';

    // 使用属性值，构建 HTML，并加入到 summaryLabel 中
    var photoHtml = feature.attributes.PHOTO || '<P>Photo not available</P>';
    var titleHtml = '<p style="font-size:18px"><b>' + featureName + '</b></p>';
    var descripHtml = '<p>' + country + '国家' + featureName + '地铁系统，' + year + '年投入运行，当前每
天服务于' + passengers + '名乘客。该网络系统包含' + stations + '站点，总里程超过' + length + '公里。</p>';
    var readmoreHtml = '<p><a href="' + link + '">更多信息...</a></p>';
    document.getElementById('summaryLabel').innerHTML = titleHtml + descripHtml + readmoreHtml;
    document.getElementById('metroImage').innerHTML = photoHtml;
}
```

（9）处理要素取消选择事件。

加入如下代码，用于处理要素取消选择事件。

```
// 处理要素取消选择事件
```

```
    function onFeatureUnselect(feature) {
        // 将 summaryLabel 中的内容设置为默认状体
        document.getElementById('summaryLabel').innerHTML = '<p>在地图上单击地铁系统，获取更多信息。</p>';
        document.getElementById('metroImage').innerHTML = ''
    }
```

（10）运行并测试程序。

用 Firefox 等浏览器直接打开 ThematicMap.html 文件，然后在地图中选择某一要素，可显示该要素的详细信息，如图 8.11 所示。



图 8.11　程序运行效果

8.5　习　题

（1）Dojo 的 dojox/geo/openlayers 是基于 OpenLayers 的地图模块，阅读如下地址的两篇文章，熟悉该模块的使用。

- http://dojotoolkit.org/reference-guide/1.10/dojox/geo/openlayers.html
- http://acuriousanimal.com/blog/2012/01/23/dojo-openlayers-new-challenges/

（2）利用下载文件“Codes\Assignment08”文件夹中 internet_users_2005.json（2005 年每个国家互联网用户量）的数据，根据 value 字段的值，使用 OpenLayers 制作图 8.12 所示的范围专题图。参考代码见同文件夹中的 Choropleth.html 文件。



图 8.12　使用 OpenLayers 制作范围专题图

（3）利用下载文件“Codes\Assignment08”文件夹中 population_2005.json（2005 年各国人口数量）的数据，根据 value 字段的值，使用 OpenLayers 制作图 8.13 所示的等级符号专题图。参考代码见同一文件夹中的 ProportionalSymbol.html 文件。

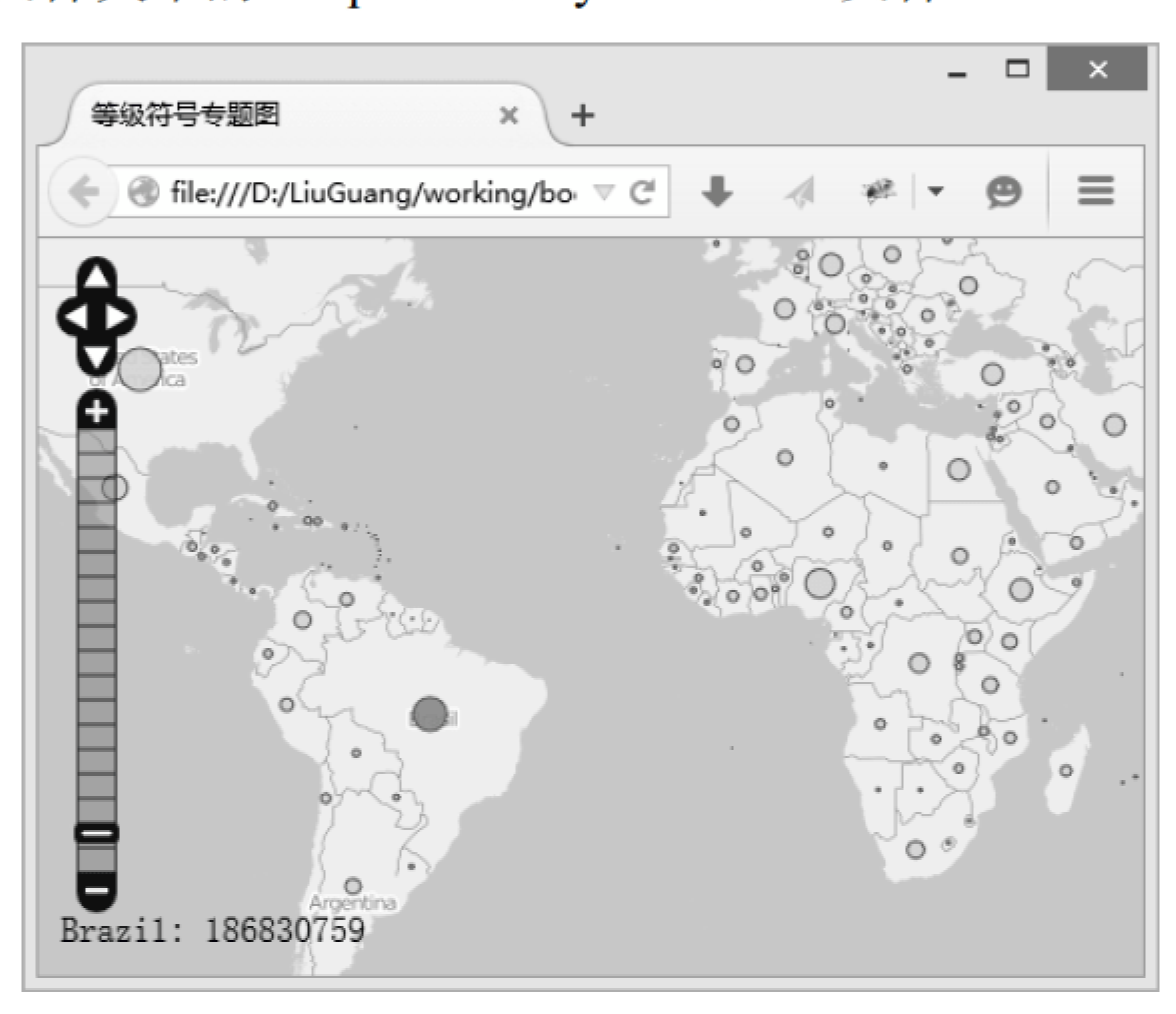


图 8.13　使用 OpenLayers 制作等级符号专题图

第 9 章

Web 要素服务

从本章可以学习到：

- ❖ 了解 WFS
- ❖ 事务性 WFS 与基于 Web 的数据编辑
- ❖ 基于 Web 的空间数据编辑功能实现

在第 7 章介绍了如何在 Web 浏览器或客户端中绘制矢量数据，主要介绍了 KML 与 GeoJSON 格式的数据。不过，还可以根据请求从 Web 服务中将数据发送到客户端。只要服务器与客户端都遵循同一规范，那么数据可以是任意格式的。为了规范通过 Web 服务发送矢量数据的过程，OGC 制定了 Web 要素服务（WFS）规范。

本章将介绍 WFS 及其服务的发布、访问与应用，并介绍如何通过该服务实现基于 Web 的空间数据编辑。

9.1　WFS

Web GIS 服务器除了能返回一张地图图像之外，也可以返回绘制该地图图像所使用的真实地理数据。用户利用这些数据可以创建他们自己的地图与应用、数据格式转换以及底层的地理操作。这类返回地理要素数据的规范称为 Web 要素服务。图 9.1 显示了 WFS 是如何将一个要素请求转换为响应的。

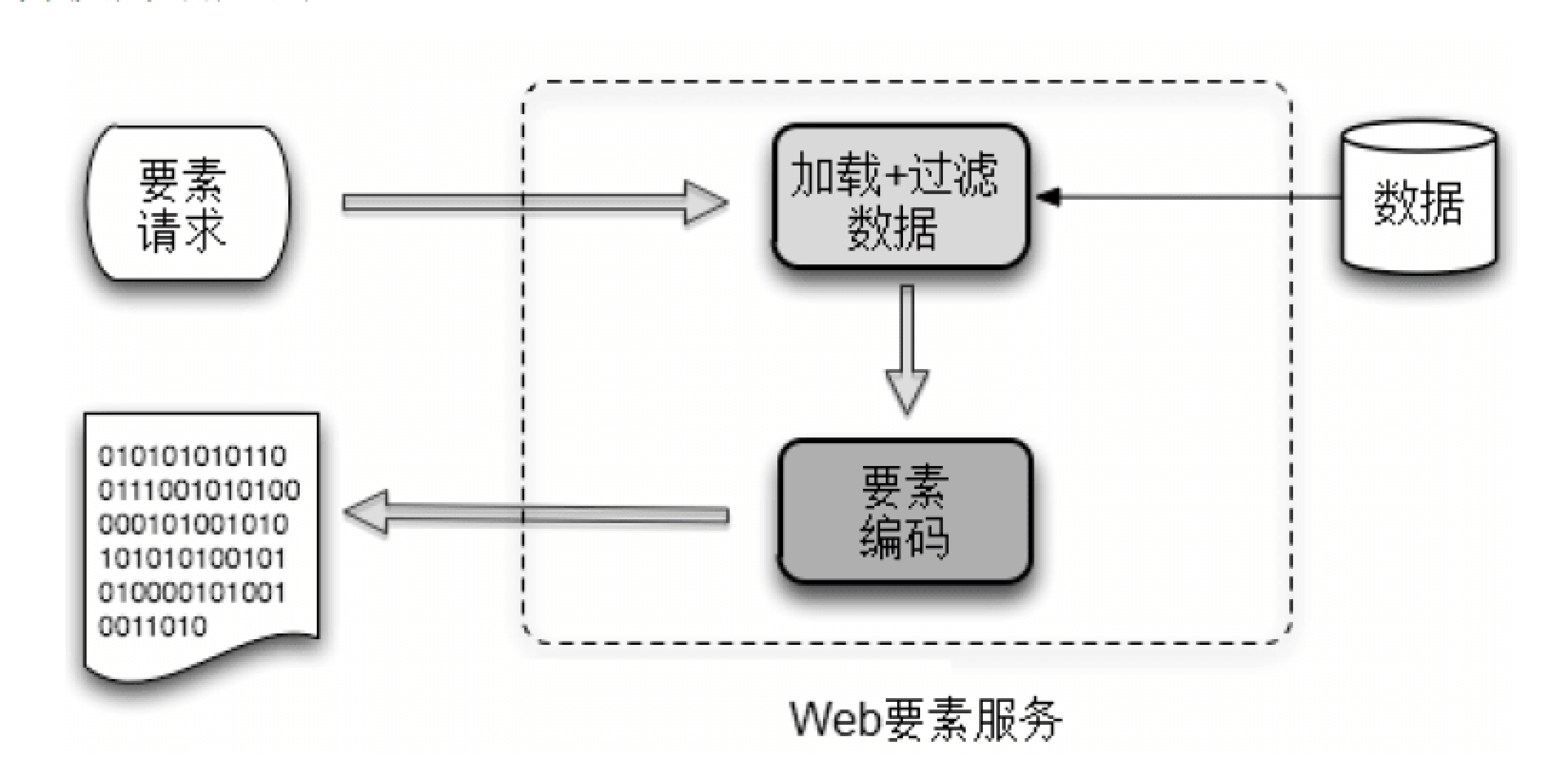


图 9.1　Web 要素服务对要素请求的响应

那么 WMS 与 WFS 有什么区别呢？WMS 是由服务器将一地图图像发送给客户端，而 WFS 是服务器将矢量数据发送给客户端，也就是在使用 WMS 时地图由服务器绘制，在使用 WFS 时地图由客户端绘制。

9.1.1　WFS 请求与响应的格式

与 WMS 类似，WFS 支持直接在 URL 地址中加参数方式的几个操作，这些操作包括 GetCapabilities、DescribeFeatureType 与 GetFeature。其中 GetFeature 操作用于获取要素数据。

下面是在 WFS 中 GetFeatures 操作的例子，该请求用于获取美国科罗拉多州要素数据：

http://suite.opengeo.org/geoserver/wfs?service=wfs&version=1.1.0&request=GetFeature&typename=usa:states&featureid=states.23

在上述请求中，service=wfs 表示使用 WFS 服务，version=1.1.0 表示使用 1.1.0 版本，

request=GetFeature 表示执行 GetFeature 操作，typename=usa:states 表示针对的是 GeoServer 服务器中 USA 工作区的名为 states 的图层，featureid=states.23 表示需要获取的要素的 ID 为 23。

WFS 使用地理标记语言（Geography Markup Language，GML）返回数据。GML 可以同时包含图形与属性信息。由于 GML 是基于 XML 的，因此比 GeoJSON 要冗长得多。

仔细查看上述请求返回的结果，可在其中找到以下图形数据：

```
<gml:posList>37.48468400000007 -109.04348799999995 38.164690000000064 -109.04176199999989 38.27548899999999 -109.06006199999996 41.0006590000001 -109.05007599999999 41.00235900000007 -102.051717 36.993015999999955 -102.04208899999992 36.99908400000004 -109.0452229999999 37.48468400000007 -109.04348799999995</gml:posList>
```

同时也可看到如下所示的属性数据：

```
<usa:NAME10>Colorado</usa:NAME10>
<usa:ALAND10>2.68431246426E11</usa:ALAND10>
<usa:AWATER10>1.170101258E9</usa:AWATER10>
<usa:DP0010001>5029196</usa:DP0010001>
<usa:DP0010020>2520662</usa:DP0010020>
<usa:DP0010039>2508534</usa:DP0010039>
```

上述类型的请求同样可用于使用 GeoServer 发布的服务。例如对于前面内容中发布的费城社区图层，可使用如下请求获取其中一个社区的空间数据：

http://localhost:8080/geoserver/wfs?service=wfs&version=1.1.0&request=GetFeature&typename=webgis:Neighborhoods&featureid=Neighborhoods.12

主要响应内容如下：

```
<wfs:FeatureCollection numberOfFeatures="1" timeStamp="2014-03-03T15:07:31.822Z" xsi:schemaLocation="http://localhost:8080/geoserver/webgis
http://localhost:8080/geoserver/wfs?service=WFS&version=1.1.0&request=DescribeFeatureType&typeName=webgis%3ANeighborhoods http://www.opengis.net/wfs http://localhost:8080/geoserver/schemas/wfs/1.1.0/wfs.xsd">
  <gml:featureMembers>
    <webgis:Neighborhoods gml:id="Neighborhoods.12">
      <webgis:the_geom>
        <gml:MultiSurface srsDimension="2" srsName="urn:x-ogc:def:crs:EPSG:3857">
          <gml:surfaceMember>
            <gml:Polygon srsDimension="2">
              <gml:exterior>
                <gml:LinearRing srsDimension="2">
                  <gml:posList>-8363968.786751106 4869301.13520122 -8363706.077778376 4871057.31164155 -8363880.846283749 4871132.918517317 -8363697.377540309 4872031.511981935 -8363780.660729433 4872179.806916264 -8363847.448310932 4872208.890548547 -8363802.926044645 4872557.878939522 -8363802.44449278 4872626.491915396 -8363025.915000884 4872530.247301338 -8361543.138729884 4872310.6731403675 -8361453.88028348 4872223.294811407 -8361493.045963939
```

```
4872015.489274301    -8361627.94355705    4871826.7318475135    -8361690.687270048    4871673.398417745
-8361627.94355705    4871403.748827802    -8361286.901117077    4870791.777211798    -8361326.368936536
4870458.7113405885    -8361498.408149585    4869986.8871721085    -8361555.111808623    4869831.380121785
-8362695.297708079    4869623.850560427    -8363168.406381819    4869548.2551895585    -8363968.786751106
4869301.13520122</gml:posList>
                              </gml:LinearRing>
                           </gml:exterior>
                        </gml:Polygon>
                     </gml:surfaceMember>
                  </gml:MultiSurface>
               </webgis:the_geom>
               <webgis:STATE>PA</webgis:STATE>
               <webgis:COUNTY>Philadelphia</webgis:COUNTY>
               <webgis:CITY>Philadelphia</webgis:CITY>
               <webgis:NAME>Olney</webgis:NAME>
               <webgis:REGIONID>214146.0</webgis:REGIONID>
            </webgis:Neighborhoods>
         </gml:featureMembers>
      </wfs:FeatureCollection>
```

9.1.2　WFS 服务器与客户端

虽然 WFS 的请求与响应的语法初看起来有些吓人，不过在实际使用过程中，并不需要我们手工来编写。GIS 软件中通常都支持查看与发布 WFS 服务。

正如上面所演示的，GeoServer 可将图层发布为 WFS 服务，而且这是默认设置，并不需要用户额外的配置。其他开源 GIW 服务器软件，例如 MapServer 与 Degree 等也都支持创建 WFS 服务。

在商业软件领域，ESRI 的 ArcGIS Server 也可用于发布 WFS 服务。不过，ESRI 开发了其自身基于 REST 的“要素服务”，用于完成 WFS 相同功能。ESRI 的 ArcGIS API for JavaScript 及其编辑控件使用的是其自身的要素服务，而不是标准的 WFS。

对于大多数的 Web 地图 API，通常将 WFS 作为一图层。例如，在 OpenLayers 中，要加入一个 WFS 服务，与前面内容介绍的加入 KML 与 GeoJSON 数据一样，使用 OpenLayers.Layer.Vector 即可，只是协议不同。对于要素的样式与符号，与前面内容介绍的也完全一样。具体如下：

```
// 定义一样是
styleMap = new OpenLayers.StyleMap({
    strokeColor: "black",
    strokeWidth: 2,
    strokeOpacity: 0.5,
```

```
        fillOpacity: 0.2
});
// 创建 WFS 图层
var wfs = new OpenLayers.Layer.Vector("States", {
        strategies: [new OpenLayers.Strategy.BBOX()],
        protocol: new OpenLayers.Protocol.WFS({
                version: "1.0.0",
                srsName: "EPSG:4326",
                url:   "http://demo.opengeo.org/geoserver/wfs",
                featureType: "states",
                featureNS: "http://www.openplans.org/topp"
        }),
        styleMap: styleMap
});
map.addLayer(wfs);
```

不过 Leaflet 以及许多其他轻量级的开源 Web 地图 API 中，并不原生支持 WFS。

客户端 GIS 软件一般都能查看 WFS 服务。

9.2 事务性 WFS 与基于 Web 的数据编辑

WFS 规范同时定义了要素编辑的规则，这就为基于 Web 进行矢量数据编辑打开了大门。通过 WFS 服务对源数据库中的数据进行更改称为事务性 WFS 或 WFS-T。

一旦启用了事务功能，WFS 客户端便可使用事务性 WFS 方法对地理数据库中的数据进行更改。下面是如何应用更改的示例：

（1）WFS 客户端连接到启用事务的已发布 WFS 服务。

（2）在服务器上锁定所编辑的要素和行（可使用 WFS 的 GetFeatureWithLock 请求执行此操作）。

（3）在 WFS 客户端上使用 WFS 编辑器执行编辑。

（4）随后在服务器上应用编辑（可使用事务性 WFS 方法执行此操作）。

当提交编辑内容后，将解除锁定并且要素可由其他 WFS 编辑器进行编辑。如果时间超时，锁定同样也可解除。可以通过使用 GetFeatureWithLock 方法指定一个超时分钟数来调整锁定时间。

插入事务不要求锁定要素。因为现有要素不能被修改（更新或删除），所以不必调用 GetFeatureWithLock。任何要求更新或删除的事务请求必须有锁定 ID。

事务性 WFS 可用来增加、删除或修改加载的要素，最重要的是能将该操作提交并保存到数据源中。因此，要修改的数据源必须存储在 PostgreSQL 这类空间数据库中，而不能是 Shapefile 等文件中。

9.3　实践 14：基于 Web 的空间数据编辑功能实现

下面将介绍如何使用 GeoServer 发布 PostgreSQL 中的数据，以及如何通过 OpenLayers 实现基于 Web 的数据编辑。

9.3.1　发布服务

（1）准备数据。

正如前面介绍的，要基于 Web 编辑矢量数据，那么其源数据应位于数据库中。

本实践使用的是“3.6 实践 5：PostGIS 的安装与初步使用”中导入到 PostgreSQL 数据库中的 Vancouver 数据图层。所以对于没有完成该实践的读者，请按照介绍的步骤将数据导入到 PostgreSQL 数据库中。

（2）添加新的数据存储。

启动 GeoServer，登录进入 GeoServer 的 Web 管理页面。在页面的左边单击“数据”中的“数据存储”，然后单击“添加新的数据存储”，进入“新建数据源”页面。在该页面中选择“PostGIS”，进入“新建矢量数据源”页面。

在“新建矢量数据源”页面中，将工作区选择为 webgis，将数据源名称设置为“PostGIS_Dataset”。在数据库连接参数部分，保留 host 与 port 的默认设置（如果不是默认安装，则需要更改），将 database 设置为 Vancouver，user 设置为 postgres，passwd 设置为 postgres 超级用户对于的密码（如果读者是完全按照本书的设置，那么密码也是 postgres）。其他都使用默认设置。最后单击页面底部的“保存”按钮。

（3）新建 PostGIS 图层。

当在“新建矢量数据源”页面中单击“保存”按钮之后，将进入“新建图层”页面。在该页面中列出了 PostGIS_Dataset 数据存储对应的 Vancouver 数据库中的所有图层，其中有一个为 vancouver 的图层。单击“发布”，进入“编辑图层”页面。

与将 Shapefile 文件发布为服务不同的是，在发布 PostGIS 数据库中的图层时，默认已经设置好了坐标参照系统，我们只需单击“从数据中计算”与“Compute from native bounds”计算并自动填充边框坐标即可。最后单击页面底部的“保存”按钮。保存以后，便可在图层列表中列出图 9.2 所示的新图层。

☐	类型	工作区	存储	图层名称	启用?	Native SRS
☐		webgis	philadelphia	city_limits	✔	EPSG:3857
☐		webgis	philadelphia	FarmersMarkets	✔	EPSG:3857
☐		webgis	philadelphia	Neighborhoods	✔	EPSG:3857
☐		webgis	philadelphia	roads	✔	EPSG:3857
☐		webgis	PostGIS_Dataset	vancouver	✔	EPSG:26910

图 9.2　将数据库中的数据发布为图层

9.3.2 基于 Web 编辑功能开发

（1）页面设计。

新建一个名为 Walkthrough14 的文件夹，在其中加入名为 WebEditor.html 的文件。该文件的内容如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>实践 14：数据编辑</title>
    <style>
        .customEditingToolbar {
            float: right;
            right: 0px;
            height: 30px;
            width: 200px;
        }
        .customEditingToolbar div {
            float: right;
            margin: 5px;
            width: 24px;
            height: 24px;
        }
        .olControlNavigationItemActive {
            background-image: url("img/editing_tool_bar.png");
            background-repeat: no-repeat;
            background-position: -103px -23px;
        }
        .olControlNavigationItemInactive {
            background-image: url("img/editing_tool_bar.png");
            background-repeat: no-repeat;
            background-position: -103px -0px;
        }
        .olControlDrawFeaturePolygonItemInactive {
            background-image: url("img/editing tool bar.png");
            background-repeat: no-repeat;
            background-position: -26px 0px;
```

```
            }
            .olControlDrawFeaturePolygonItemActive {
                background-image: url("img/editing_tool_bar.png");
                background-repeat: no-repeat;
                background-position: -26px -23px;
            }
            .olControlModifyFeatureItemActive {
                background-image: url("img/move_feature_on.png");
                background-repeat: no-repeat;
                background-position: 0px 1px;
            }
            .olControlModifyFeatureItemInactive {
                background-image: url("img/move feature off.png");
                background-repeat: no-repeat;
                background-position: 0px 1px;
            }
            .olControlDeleteFeatureItemActive {
                background-image: url("img/remove_point_on.png");
                background-repeat: no-repeat;
                background-position: 0px 1px;
            }
            .olControlDeleteFeatureItemInactive {
                background-image: url("img/remove_point_off.png");
                background-repeat: no-repeat;
                background-position: 0px 1px;
            }
        </style>
        <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
        </script>
        <script></script>
    </head>
    <body onload="init()">
        <div id="map" style="width:500px; height:500px;"></div>
        <div id="coordinates"></div>
        <div id="nodelist"></div>
    </body>
    </html>
```

在页面的样式中用到了 img 文件夹中的 editing_tool_bar.png 与 move_feature_on.png 等 5 个图片，这些图片可从下载文件“Codes\Walkthrough14\img”文件夹中找到。

（2）初始化地图并加入 WFS 图层。

在<script>与</script>之间加入如下 JavaScript 代码：

```
function init() {
    var WGS84 = new OpenLayers.Projection("EPSG:4326");
    var WGS84_google_mercator = new OpenLayers.Projection("EPSG:900913");

    var map = new OpenLayers.Map("map", {
        controls: [
            new OpenLayers.Control.Navigation(),
            new OpenLayers.Control.PanZoom(),
            new OpenLayers.Control.LayerSwitcher(),
            new OpenLayers.Control.MousePosition({
                div: document.getElementById("coordinates")
            })
        ],
        displayProjection: WGS84
    });
    var mapextent = new OpenLayers.Bounds(-123.17341, 49.24343, -123.06183, 49.29899).transform
(WGS84, WGS84_google_mercator);

    // 底图
    var openstreetmap = new OpenLayers.Layer.OSM();

    // 设置保存策略
    var saveStrategy = new OpenLayers.Strategy.Save();
    saveStrategy.events.register("success", '', showSuccessMsg);
    saveStrategy.events.register("failure", '', showFailureMsg);

    // WFS-T 可编辑图层
    var wfs_layer = new OpenLayers.Layer.Vector("Editable Features", {
        strategies: [new OpenLayers.Strategy.BBOX(), saveStrategy],
        protocol: new OpenLayers.Protocol.WFS({
            version: "1.1.0",
            // 数据加载的 URL 路径
            url: "http://localhost:8080/geoserver/wfs",
            // 工作区关联的命名空间 URI
            featureNS: "http://localhost:8080/geoserver/webgis",
            maxExtent: mapextent,
            // 图层名称
            featureType: "vancouver",
            // 空间坐标所在字段名
            geometryName: "geom",
            schema: "http://localhost:8080/geoserver/wfs/DescribeFeatureType?version=1.1.0&;typename=
webgis:vancouver"
```

```
            })
        });

        map.addLayers([openstreetmap, wfs_layer]);
        map.zoomToExtent(mapextent);
    }

    function showMsg(szMessage) {
        document.getElementById("nodelist").innerHTML = szMessage;
        setTimeout("document.getElementById('nodelist').innerHTML = ''", 2000);
    }

    function showSuccessMsg() {
        showMsg("事务成功执行");
    };

    function showFailureMsg() {
        showMsg("在执行事务过程中出现错误");
    };
```

地图初始化代码与前面所有示例基本都一致，在添加 WFS 图层时，为了实现编辑的保存，在设置策略时增加了一个保存策略。

在设置 WFS 协议时，其 url 参数指定了数据加载 URL 路径，在程序运行时 OpenLayers 向该地址发出 POST 请求。使用 Firebug 或其他开发者工具，可以查看这些请求。例如图 9.3 就是使用 Firebug 查看到的加载 WFS 数据时 OpenLayers 发送的 POST 请求。

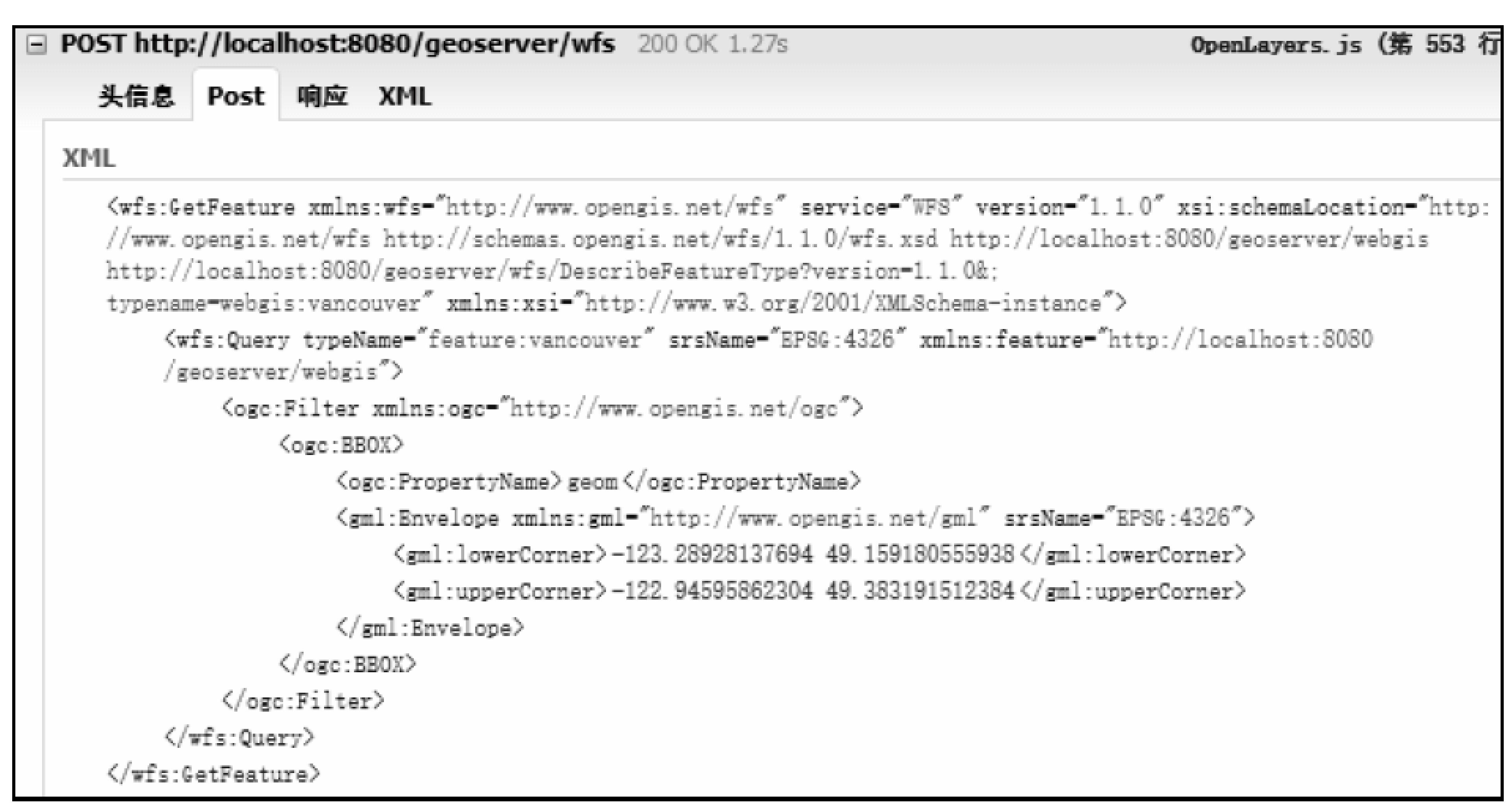


图 9.3　利用 Firebug 查看 OpenLayers 获取 WFS 数据时发送的请求

在设置 WFS 协议时，其 featureNS 参数指定的是要素图层所在工作区对应的命名空间

URI。该 URI 是指创建工作区时比较随意设置的，因此没有一定之规。例如 GeoServer 自带的 cite 工作区，其关联的命名空间 URI 为“http://www.opengeospatial.net/cite”，而与 sde 工作区关联的是“http://geoserver.sf.net”。因此，在开发时需要仔细确认。可以在 GeoServer 的 Web 管理页面，查看并重新设置某个工作区对应的命名空间 URI。例如图 9.4 显示的是我们增加的 webgis 工作区相关设置。



图 9.4　查看某个工作区关联的命名空间 URI

在设置 WFS 协议时，其 featureType 参数指定的是图层名称，而 schema 中的 typename 参数应同时包含工作区名称与图层名称。此外，geometryName 参数指定的是包含空间坐标信息的字段名称。该字段名称对于不同的图层可能会存在差异。可以在 GeoServer 的 Web 管理页面的图层列表中选择某图层，进入“编辑图层”页面，在“数据”选项卡中的底部可以查看空间坐标对应的字段名称。例如图 9.5 显示的是我们使用的 vancouver 图层中所包含的字段，通过分析可以得知空间坐标对应的字段名称是“geom”。



图 9.5　确定包含空间坐标字段的名称

（3）增加编辑工具条。

在 init()函数中，再加入如下代码：

```
// 增加自定义编辑工具条
var panel = new OpenLayers.Control.Panel({
    'displayClass': 'customEditingToolbar'
```

```
});

var navigate = new OpenLayers.Control.Navigation({
    title: "平移地图"
});

var draw = new OpenLayers.Control.DrawFeature(
    wfs_layer,
    OpenLayers.Handler.Polygon, {
        title: "绘制多边形",
        displayClass: "olControlDrawFeaturePolygon",
        multi: true
    }
);

var edit = new OpenLayers.Control.ModifyFeature(wfs_layer, {
    title: "修改多边形",
    displayClass: "olControlModifyFeature"
});

var del = new DeleteFeature(wfs_layer, {
    title: "删除要素"
});

var save = new OpenLayers.Control.Button({
    title: "保存",
    trigger: function () {
        if (edit.feature) {
            edit.selectControl.unselectAll();
        }
        saveStrategy.save();
    },
    displayClass: "olControlSaveFeatures"
});

// 在面板中加入按钮控件
panel.addControls([navigate, save, del, edit, draw]);
// 设置默认的控件
panel.defaultControl = navigate;
// 将面板控件加入到地图中
map.addControl(panel);
```

（4）实现删除要素按钮控件。

平移地图、绘制多边形以及修改多边形，OpenLayers 都提供了对应的控件。对于保存修改结果，也只需要使用一个按钮控件，并在其中调用保存策略对象的保存函数，便可实现保存功能。而对于删除要素，则需要编写相对多一点的代码。需要我们从 OpenLayers 的控件基础类来扩展。在 showFailureMsg()函数的下面，加入如下代码，用于实现删除要素按钮控件。

```
// 设置删除要素工具
var DeleteFeature = OpenLayers.Class(OpenLayers.Control, {
    initialize: function (layer, options) {
        OpenLayers.Control.prototype.initialize.apply(this, [options]);
        this.layer = layer;
        this.handler = new OpenLayers.Handler.Feature(
        this, layer, { click: this.clickFeature }
        );
    },
    clickFeature: function (feature) {
        // 如果要素不包含 fid
        if (feature.fid == undefined) {
            this.layer.destroyFeatures([feature]);
        } else {
            feature.state = OpenLayers.State.DELETE;
            this.layer.events.triggerEvent("afterfeaturemodified", {
                feature: feature
            });
            feature.renderIntent = "select";
            this.layer.drawFeature(feature);
        }
    },
    setMap: function (map) {
        this.handler.setMap(map);
        OpenLayers.Control.prototype.setMap.apply(this, arguments);
    },
    CLASS_NAME: "OpenLayers.Control.DeleteFeature"
});
```

（5）运行与调试。

用 Firefox 等浏览器直接打开 WebEditor.html 文件，然后在地图中便可增加、修改与删除多边形要素，单击“保存”按钮可将编辑结果提交到服务器，更改数据库中的数据，运行效果如图 9.6 所示。程序代码位于下载文件的“Codes\Walkthrough14”文件夹中。

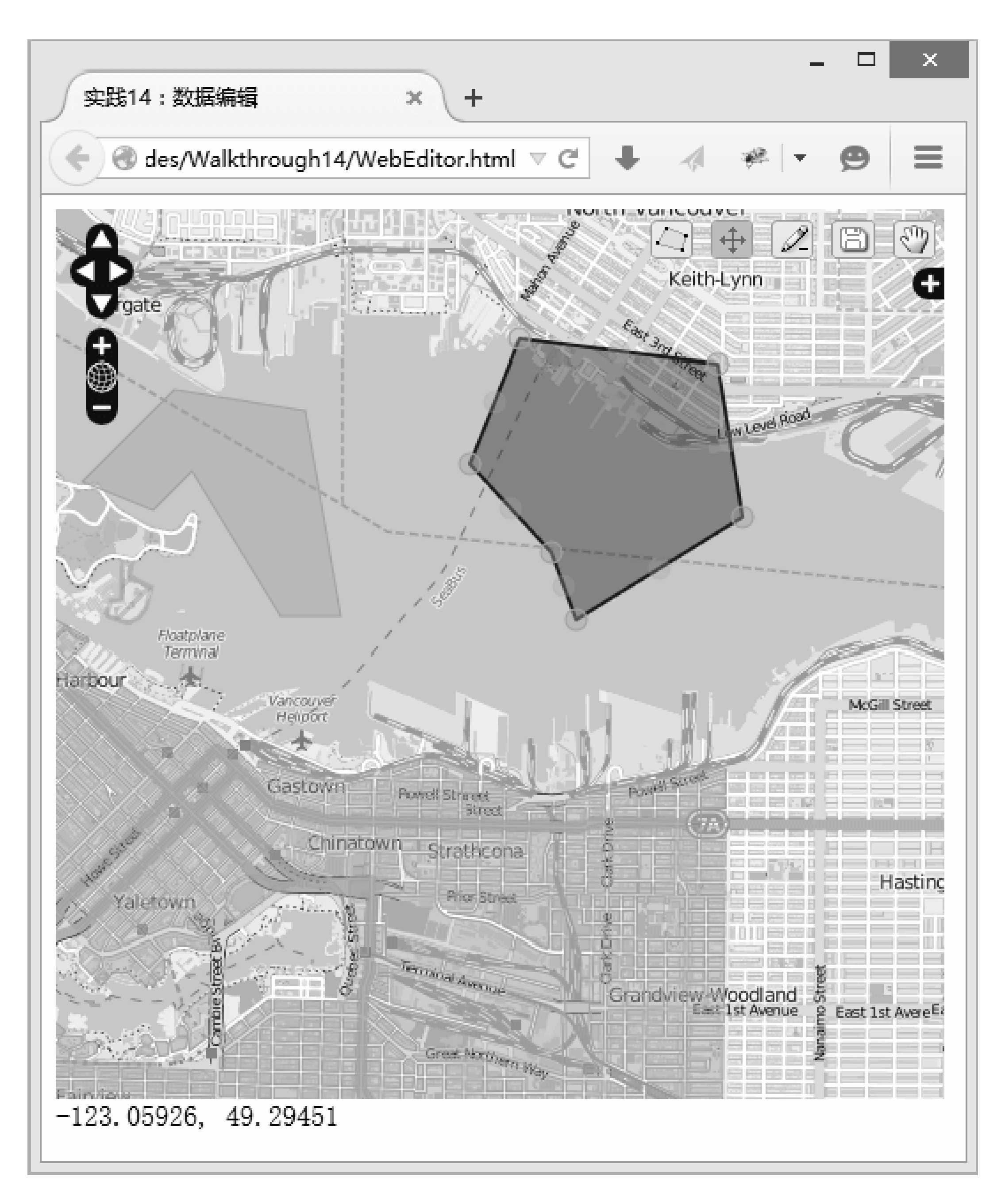


图 9.6　程序运行效果

如果程序在运行中没有成功加载 vancouver 图层，可用 Firebug 来查看原因。如果出现类似图 9.7 所示的异常响应结果，那么表示在设置 WFS 协议时工作区关联的命名空间 URI 或图层名称设置出现了错误。



图 9.7　命名空间 URI 或图层名称错误时服务器返回的异常响应

如果异常响应中包含如下类似的信息，则表示空间坐标对应的字段名设置错误了。

```
<ows:Exception exceptionCode="InvalidParameterValue">
    <ows:ExceptionText>Illegal property name: the_geom for feature type webgis:vancouver</ows:ExceptionText>
</ows:Exception>
```

对于这类错误，可按照前面介绍的方法，通过仔细查看 GeoServer 的 Web 管理页面中的信息而进行改正。

9.4 习 题

（1）利用下载文件“Codes\Assignment09”文件夹中的 postal_code_fsa.zip，将其中的 Shapefile 数据发布为服务，并编码对其进行符号化与标注，使得运行效果如图 9.8 所示。



图 9.8　对 WFS 图层进行符号化及标注

（2）利用地址为“http://demo.opengeo.org/geoserver/wfs/DescribeFeatureType?version=1.1.0&typename=og:roads”的 WFS 服务以及 OpenLayers.Control.Snapping 与 OpenLayers.Control.Split 两控件，编码实现数据编辑时的线捕捉与打断功能，运行效果如图 9.9 所示。参考代码见下载文件“Codes\Assignment09\Wfs-Snap-Split.html”文件。



图 9.9　数据编辑过程中线的捕捉与打断功能

第 10 章

WCS 及多维数据

从本章可以学习到：

- WCS 及其操作
- 多维数据与图像镶嵌插件
- 多维数据 WCS 的发布
- 在 OpenLayers 中访问 WCS

在第 9 章介绍的 WFS 是在 Web 上发布矢量数据服务，而对于栅格数据服务，OGC 制定的则是 WCS（Web Coverage Services，网络覆盖服务）。WCS 服务所返回的数据可作为分析和建模操作的输入参数。这与 OGC WMS 服务形成鲜明对比，后者仅返回数据的图片。通过 WCS 服务获取的栅格数据集被称为覆盖。不要将此覆盖与 ArcGIS 先前版本中所提供的矢量数据集（也称为覆盖）相混淆。

本章将介绍 WCS 服务规范及其在多维数据中的应用，以及如何利用 GeoServer 将带有时间与高程信息的多维数据发布为 WCS 服务。

10.1 WCS 及其操作

WCS 是由 OGC 创建的用于在 Web 上共享覆盖地理信息的开放规范。这里的覆盖意指表示空间变化现象的地理信息。可以把 WCS 看作为栅格数据的 Web 要素服务。它是以原始图像方式获取地图的“源代码”，而不是原始矢量数据方式。

虽然 WCS 与 WMS 从服务器上返回的都是以图像，但是它们之间一个重要的区别是 WCS 能够返回更多的详细信息，包括有价值的元数据和更多的格式，返回的数据可作为分析和建模操作的输入参数。此外，如果 WCS 是一个多维格式，那么可实现更为精确的查询。

WCS 服务支持以下操作：

（1）请求服务级别元数据和数据的简要描述（GetCapabilities）。

（2）请求一个或多个覆盖的完整描述（DescribeCoverage）。

（3）以熟知的格式请求覆盖（GetCoverage）。

10.1.1 GetCapabilities 操作

GetCapabilities 操作用于得到 WCS 服务器所支持的操作和服务（能力）的列表。

一个典型的 WMS 服务的 GetCapabilities 请求使用如下的地址（针对 http://www.example.com/wcs 地址）：

http://www.example.com/wcs?service=WCS&request=GetCapabilities

例如对于本地计算机安装的 GeoServer，其请求地址如下：

http://localhost:8080/geoserver/ows?service=WCS&request=GetCapabilities

由于许多 WCS 支持多个版本，事实上当前许多服务同时支持 WCS 1.0.0 与 1.1.x，甚至是 2.0.x，因此如果在请求地址中没有包含版本参数，那么得到的是可获取的最新版本的信息。通过查看此信息，便可得到服务支持的所有版本信息。例如上述地址返回的响应中包含如下描述所支持的版本信息：

```
<ows:ServiceTypeVersion>2.0.1</ows:ServiceTypeVersion>
<ows:ServiceTypeVersion>1.1.1</ows:ServiceTypeVersion>
<ows:ServiceTypeVersion>1.1.0</ows:ServiceTypeVersion>
```

而对于特点版本，可在地址中通过 version 参数中指定版本，例如：

http://localhost:8080/geoserver/ows?service=WCS&version=1.1.1&request=GetCapabilities

在返回的 GetCapabilites 文档中，OperationsMetadata 部分包含了服务支持的操作。

GetCapabilites 文档中 Contents 部分包含了各个覆盖的 ID 以及其他一些信息，例如所包含的波段数。下面是本地 GeoServer 返回的 GetCapabilites 文档 Contents 部分中一个覆盖的信息：

```
<wcs:CoverageSummary>
  <wcs:CoverageId>nurc   mosaic</wcs:CoverageId>
  <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubtype>
  <ows:WGS84BoundingBox>
    <ows:LowerCorner>6.346 36.492</ows:LowerCorner>
    <ows:UpperCorner>20.83 46.591</ows:UpperCorner>
  </ows:WGS84BoundingBox>
  <ows:BoundingBox crs="http://www.opengis.net/def/crs/EPSG/0/EPSG:4326">
    <ows:LowerCorner>6.346 36.492</ows:LowerCorner>
    <ows:UpperCorner>20.83 46.591</ows:UpperCorner>
  </ows:BoundingBox>
</wcs:CoverageSummary>
```

得到覆盖的 ID 后，便可执行 DescribeCoverage 操作了。要注意的是，上面的 GetCapabilites 文档是使用 2.0.1 版本请求所返回的响应，覆盖 ID 使用的是 CoverageId，而对于 1.1.1 的版本，使用的是 Identifier。在构造 DescribeCoverage 操作与 GetCoverage 操作时，对于不同版本，需要使用其对应的参数。

10.1.2　DescribeCoverage 操作

DescribeCoverage 操作允许客户端请求某个 WCS 服务的一个或多个栅格图层的全部描述信息。服务端会返回描述所请求的覆盖图层详细信息的 XML 文档。

DescribeCoverage 操作的主要请求参数（1.1.1 版本）如表 10.1 所示.

表 10.1　DescribeCoverage 操作请求主要参数（1.1.1 版本）

请求参数	是否必需	描述
service=WCS	是	服务类型
request=DescribeCoverage	是	请求名称
version	是	请求的 WCS 服务的版本
identifiers	是	指定所要请求的图层 identifiers=identifier1, identifier2……

例如对于 GeoServer 中 nurc 工作区中的 mosaic，其 DescribeCoverage 操作请求地址如下：

http://localhost:8080/geoserver/ows?service=WCS&version=1.1.1&request=DescribeCoverage&identifiers=nurc:mosaic

10.1.3 GetCoverage 操作

通过 GetCapabilities 和 DescribeCoverage 这两个操作，可以了解服务端允许哪些请求以及哪些数据是可以获取的，然后进行 GetCoverage 操作。该操作最终返回指定地理范围内指定域值内的某栅格数据。

在构造 GetCoverage 操作时，除了包含 service、request、version 与 identifiers 参数值之外，还需要包含 domainSubset 参数。该参数定义所要请求覆盖的时空范围，它又包含 boundingBox（必选）和 temporalSubset（可选）两个参数，前者指定地理范围，后者指定时间范围。此外，还需要用 output 参数指定输出设置，该参数又包括 gridCRS（返回数据的地理参照系统，可选）、format（返回数据的格式，必选）和 store（“true”表示需要服务端把返回数据的所有内容存储在一个网络位置，并返回其 URL；“false”表示需要服务端直接返回数据；二者可选）。

例如对于 GeoServer 中 nurc 工作区中的 mosaic，其 GetCoverage 操作请求地址如下：

http://localhost:8080/geoserver/ows?service=WCS&Version=1.1.1&REQUEST=GetCoverage&Identifier=nurc:mosaic&BoundingBox=36.492,6.346,46.591,20.83,urn:ogc:def:crs:EPSG::4326&Format=image/png&Store=true

服务器端接到请求后，根据客户端发送请求中的参数，以 output 中 format 指定的格式返回符合条件的数据，同时返回数据相关的元信息。例如上述请求地址返回类似如下的响应：

```
<wcs:Coverage>
  <ows:Title>mosaic</ows:Title>
  <ows:Abstract>Generated from ImageMosaic</ows:Abstract>
  <ows:Identifier>mosaic</ows:Identifier>
  <ows:Reference xlink:href="http://localhost:8080/geoserver/temp/wcs/mosaic_105888126054900.png"/>
</wcs:Coverage>
```

Reference 的 http://localhost:8080/geoserver/temp/wcs/mosaic_105888126054900.png 地址便是指定返回的覆盖数据，访问该地址，可以得到图 10.1 所示的结果。

然而在 GeoServer 的 Web 管理页面中，通过 OpenLayer 来预览 nurc:mosaic 时却得到一个坐标轴不同的图像，如图 10.2 所示。

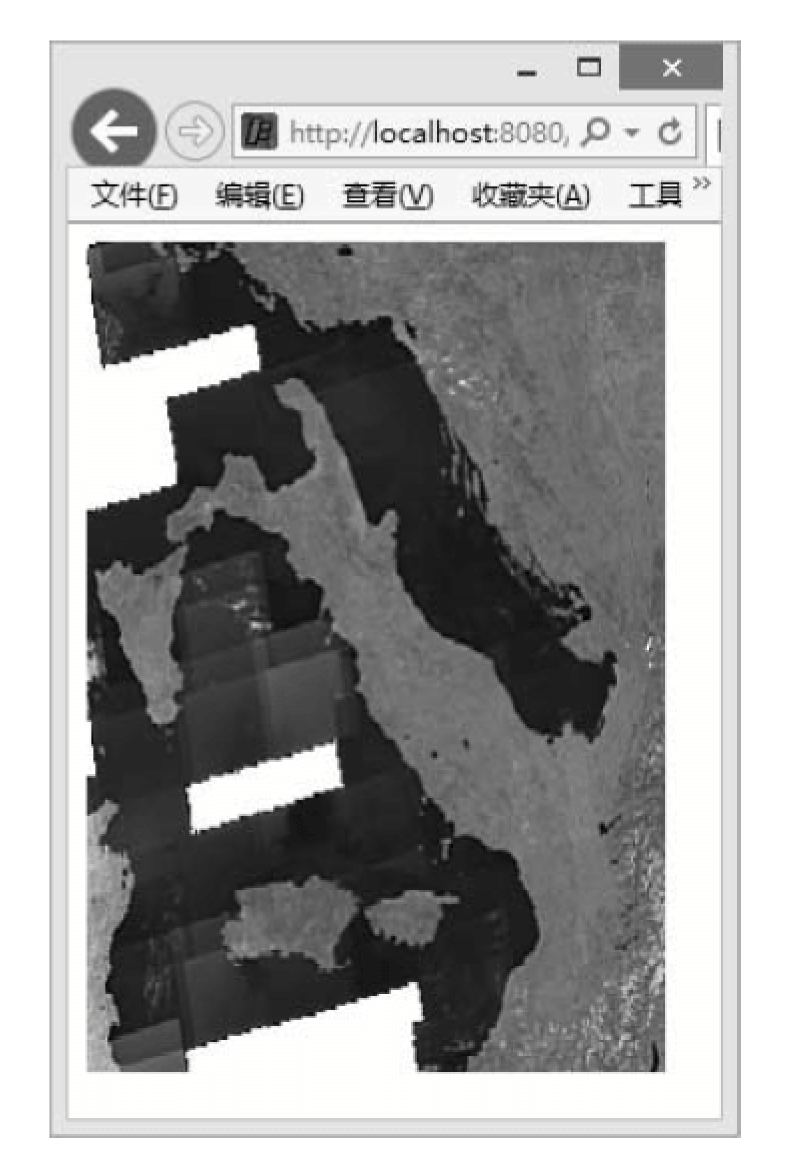


图 10.1　使用 GetCoverage 操作获取覆盖数据

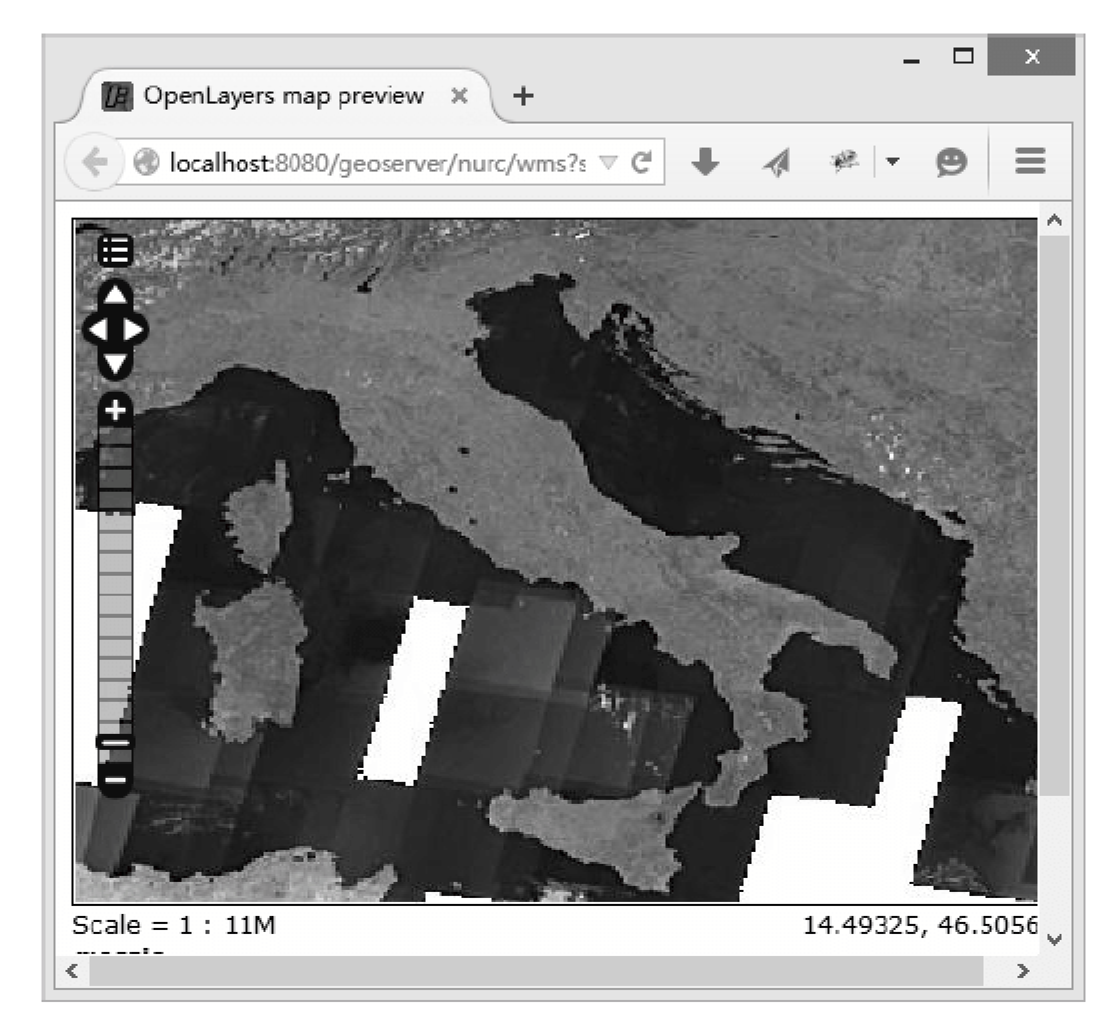


图 10.2　通过 OpenLayer 预览 nurc:mosaic

这主要还是由于历史原因造成的，具体说明请查看如下地址的说明文档：

http://docs.geoserver.org/latest/en/user/services/wfs/basics.html#axis-ordering

解决方案是在请求参数中增加 GridBaseCRS 参数，例如对于前面的请求地址，将其修改为如下地址便能得到正确的图像：

http://localhost:8080/geoserver/ows?service=WCS&Version=1.1.1&REQUEST=GetCoverage&Identifier=nurc:mosaic&BoundingBox=36.492,6.346,46.591,20.83,urn:ogc:def:crs:EPSG::4326&GridBaseCRS=EPSG:4326&Format=image/png&Store=true

而对于 2.0.1 的版本，就没有必要再这么处理了，而且请求也相对要简单。例如可使用如下 2.0.1 版本的请求地址，获取 nurc 工作区中 mosaic 覆盖，并且直接返回图像：

http://localhost:8080/geoserver/wcs?service=WCS&version=2.0.1&CoverageId=nurc__mosaic&request=GetCoverage&format=image/png

10.2　多维数据与图像镶嵌插件

地理数据，特别是遥感数据，通常是多维的。例如遥感数据通常包含多个波段的数据。在 GeoServer 中默认就可包含时间与高程维度。此外，还可以利用 GeoServer 的图像融合插件（ImageMosaic Plugin）创建多维图像。

10.2.1　多维数据

在气象和海洋领域，温度、湿度以及洋流等地球物理参数在不同的时间、深度是不一样

的，因此这些数据通常包含时间、深度以及压力等维度，如图 10.3 所示。

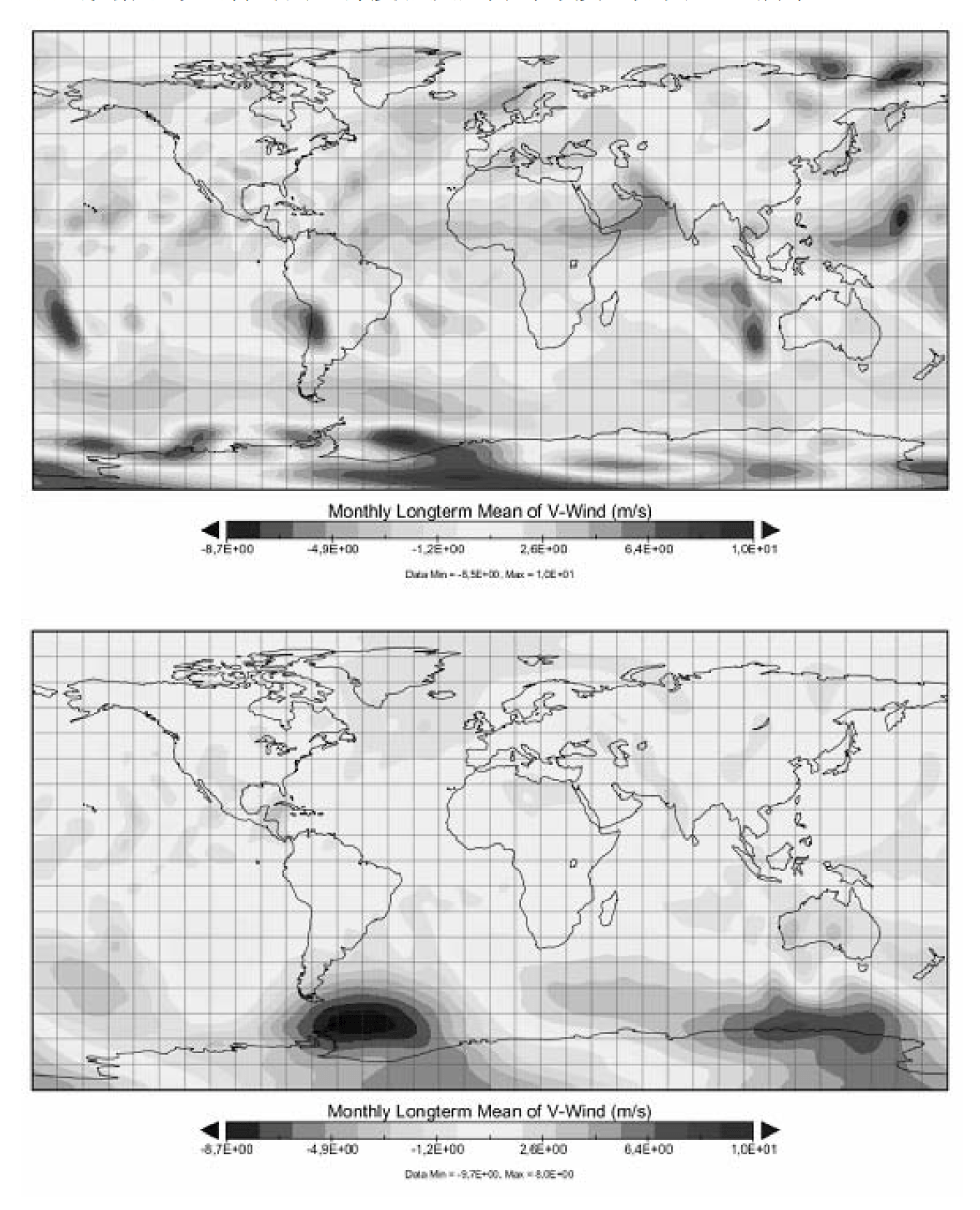


图 10.3　不同时间和不同大气压力水平下风的 V 量月平均值

现假设有某一个用于预测一系列深度下不同时间海水温度的海洋学模型，或者用于预测不同维度不同时间风力的气象学模型，可以将这些实体想象为多维数据或超立方体，其维度包含平面空间的两个维度、Z 维度（海拔/高度/深度/压力）以及时间维度，如图 10.4 所示。

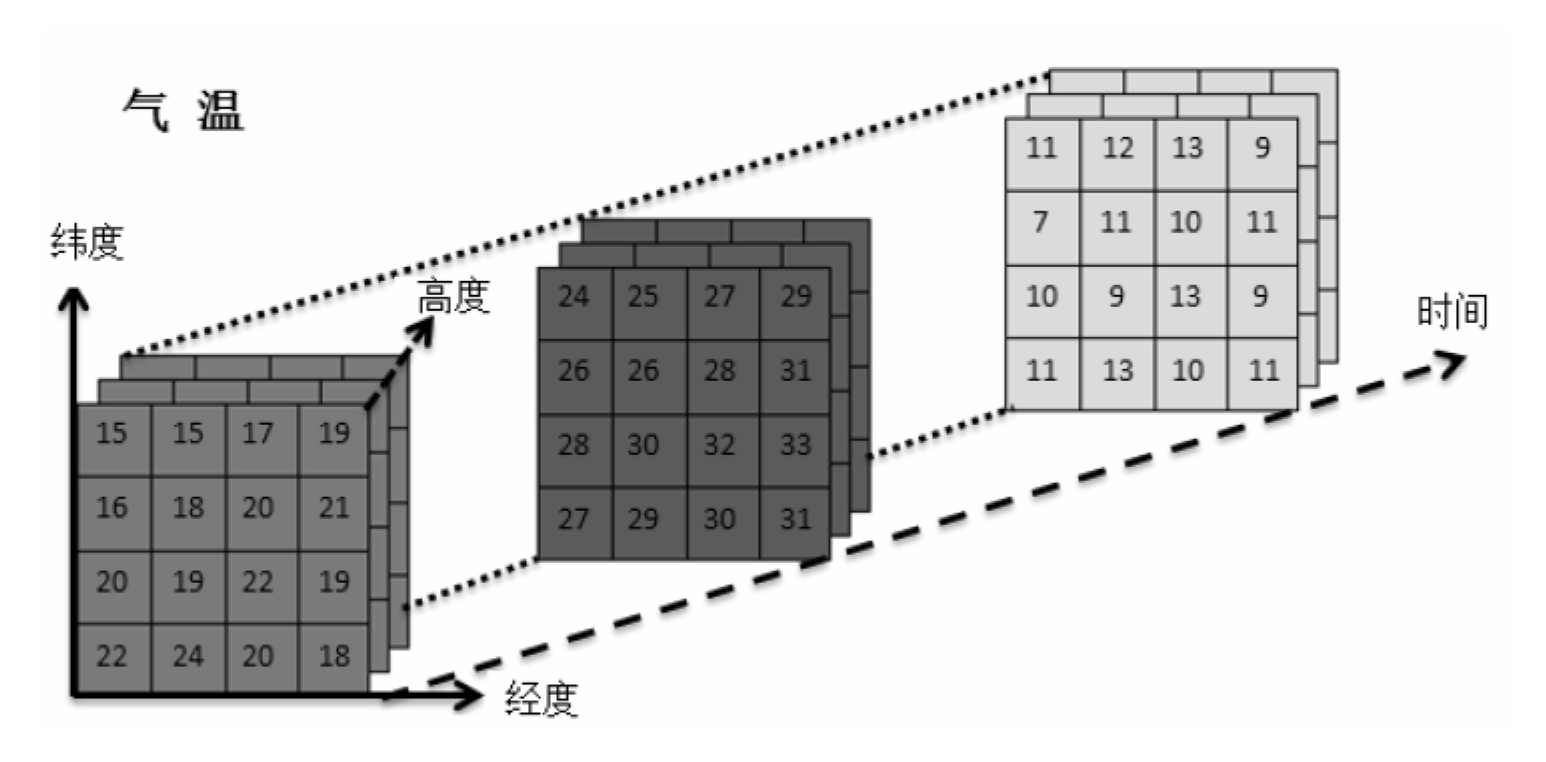


图 10.4　将气温表示为四维超立方体

类似的概念可以应用于遥感数据。卫星传感器在不同时间接收不同波长的辐射。在这种情况下，波长可以被认为是一个额外的维度。

对于这种类型的数据，需要特定的管理、处理和服务。这类数据服务在标准的二维空间维度之外，还需要识别那些额外维度（时间、高程、自定义维度），并允许用户在这些维度之间请求数据子集。

例如，用户与这些服务进行交互时，可能想要检索在某月某日按一定深度收集的当时水温数据。该请求除了空间维度之外，还包含了时间维度，如图 10.5 所示。

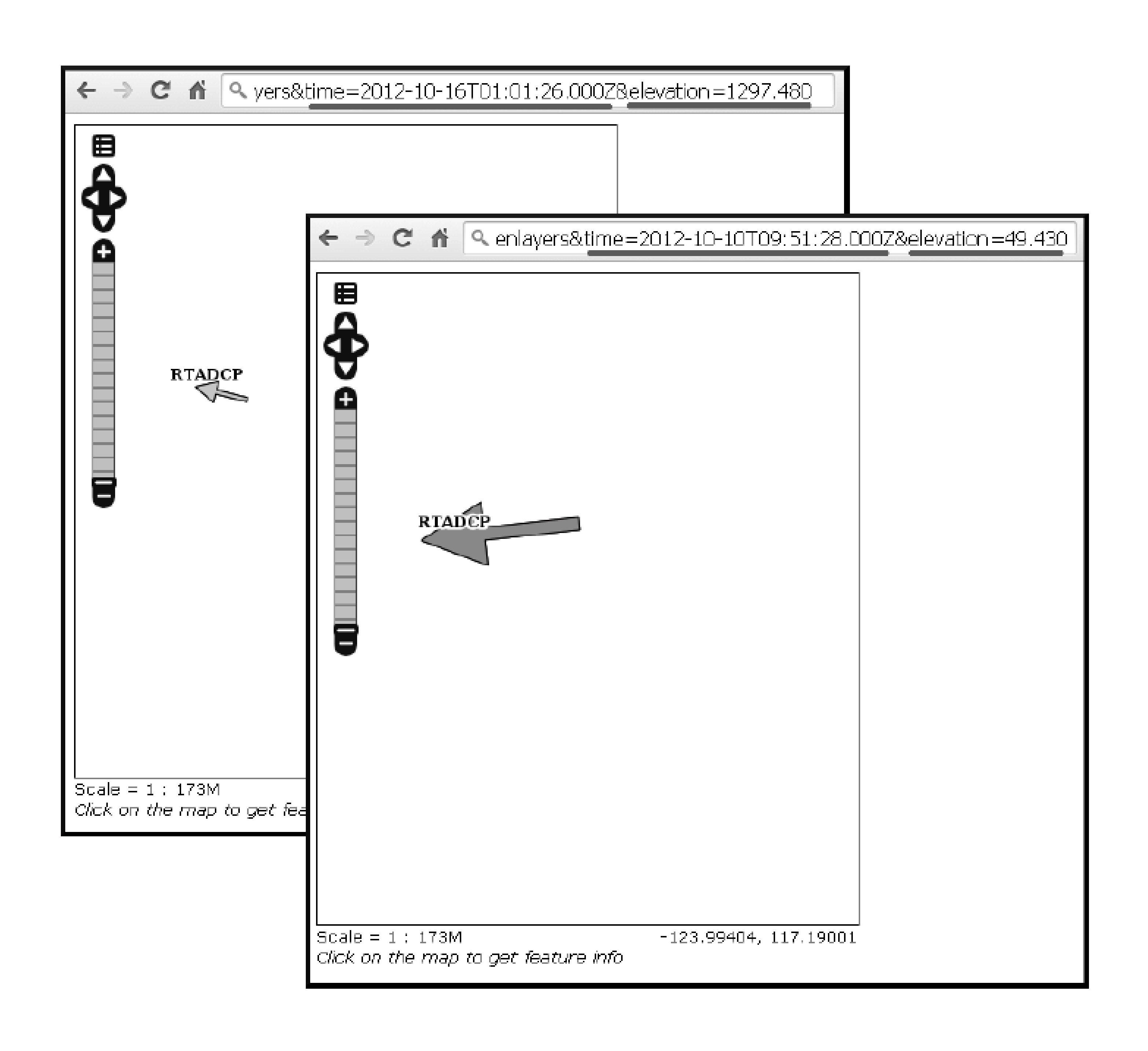


图 10.5　利用 OpenLayer 预览某一矢量数据中不同维度的数据子集

10.2.2 图像镶嵌插件

图像镶嵌是用于将一组带地理参考的栅格文件镶嵌在一块的插件，由 GeoTools 提供。该插件可用于 GeoServer 所使用的大多数格式（如 GeoTIFF）、带世界文件的栅格、一些 GDAL 支持的格式，以及 NetCDF 和 GriB 文件。

简单地说，图像镶嵌插件负责将一组类似的栅格数据归为一组，将它们作为一个栅格数据集（也可能是多维的）对外提供服务。例如，图像镶嵌插件可用于下列一些情况：

（1）将一组空间邻近的遥感图像进行镶嵌；

（2）将同一地理范围内不同时间与/或不同高程或不同维度（例如风向）的栅格数据进行镶嵌，组成一多维数据集。

在 GeoServer 中，通过指定一镶嵌根目录来设置图像镶嵌。通常，该目录中包含一组要被镶嵌的数据文件以及一组配置文件。图像镶嵌插件首先通过解析提供的配置文件，然后从根目录中去寻找镶嵌数据。镶嵌插件会检查找到的每个镶嵌数据，看其是否已经属于某一已经配置的覆盖或是一个新文件。然后它会根据数据类型、坐标参照系统、波段数等特征，更新内部的数据子集目录，以将它们组合起来。

这些组成镶嵌的一组数据与其相关信息（位置、空间范围等）保存为索引，该索引可以是一个 Shapefile 文件、一个 PostGIS 数据库、一个 Oracle 数据库或一个 H2 数据库。对于将这些信息存在数据库管理系统中还是存在一简单的 Shapefile 中，可以在一个名为 datastore.properties 的配置文件中通过设置来指定。此外，还可以通过一组辅助文件，来指定该镶嵌中可获取的维度、索引以及获取维度值的方式。我们将在实践部分再详细描述。

一旦配置了图像镶嵌，通常获取其中数据子集的方式是通过 OGC 的服务请求，例如使用 WMS 请求获取栅格数据的地图，或使用 WCS 请求获取源数据。

10.3 实践 15：多维数据 WCS 的发布

本实践将介绍如何使用图像镶嵌插件构建一个时间序列的覆盖。该覆盖保存为一可查询的结构，允许通过时间条件来得到具体的数据集。

10.3.1 发布时间序列栅格数据

（1）数据准备。

建立一个简单的文件夹，例如本实践使用“C:\Data\snowLZWdataset”。并将下载文件“Data\snow”文件夹中的 4 个文件复制进来。其中 3 个分别是 snow_20091001.tif、snow_20091101.tif 与 snow_20091201.tif 表示 3 个不同时间的栅格数据，另一个名为 snow_style.sld，是个样式文件。

（2）设置配置文件。

对于时间序列影像需要 3 个配置文件。一个是与图层名同名的.properties 文件，该文件包含将镶嵌空间索引存储到数据库中所需要的相关信息。如果是使用 Shapefile 作为索引存储，则 GeoServer 会自动生成该文件。本实践就是使用该方式，因此不需要创建。

第二个文件是 indexer.properties，该文件指定时间变量属性、高程属性的名称以及这些属性的类型。在 C:\data\snowLZWdataset 文件夹中创建一个名为 indexer.properties 的文本文件，其内容如下：

```
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Integer
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion)
```

第三个文件是 timeregex.properties，指定了从文件名提取时间信息的规则表达式。在“C:\Data\snowLZWdataset”文件夹中创建一个名为 timeregex.properties 的文本文件，其内容如下：

```
regex=[0-9]{8}
```

（3）修改 GeoServer 启动参数。

要支持时间序列图层，GeoServer 需要配置合适的时区。为了将时区设置为协调通用时间（Coordinated Universal Time，UTC），需要在加载 Java 程序时加入适当的参数。

用文本文件的方式打开 GeoServer 的启动程序 startup.bat 文件，如果是默认安装，那么该文件位于“C:\Program Files\GeoServer 2.6.2\bin”文件夹中。在该文件的最后一个参数，即“-jar "C:\Program Files\GeoServer 2.6.2\start.jar"”前面加入如下参数：

```
-Duser.timezone=GMT
```

如果使用 Shapefile 作为镶嵌索引存储，那么还需要加入如下参数，以使 Shapefile 支持时间戳：

```
-Dorg.geotools.shapefile.datetime=true
```

（4）创建图像镶嵌数据存储。

启动 GeoServer，用 admin 身份登录进入 GeoServer 的 Web 管理页面。新建一个 ImageMosaic 类型的数据存储。将工作区选择为 webgis，数据源名称设置为“snow_file_store”，URL 设置为在第一步中创建的目录，基本设置如图 10.6 所示。



图 10.6　新建 ImageMosaic 数据存储

单击“保存”按钮后，GeoServer 在数据存储对应的目录中创建用于镶嵌的相关文件（.dbf、.prj、.properties、.shp 与.shx）。如果在该目录中已经存在同名文件，则会出现错误。

（5）增加新图层。

在“添加栅格数据源”中设置了新的数据存储之后，GeoServer 的 Web 管理页面将进入图 10.7 所示的页面。

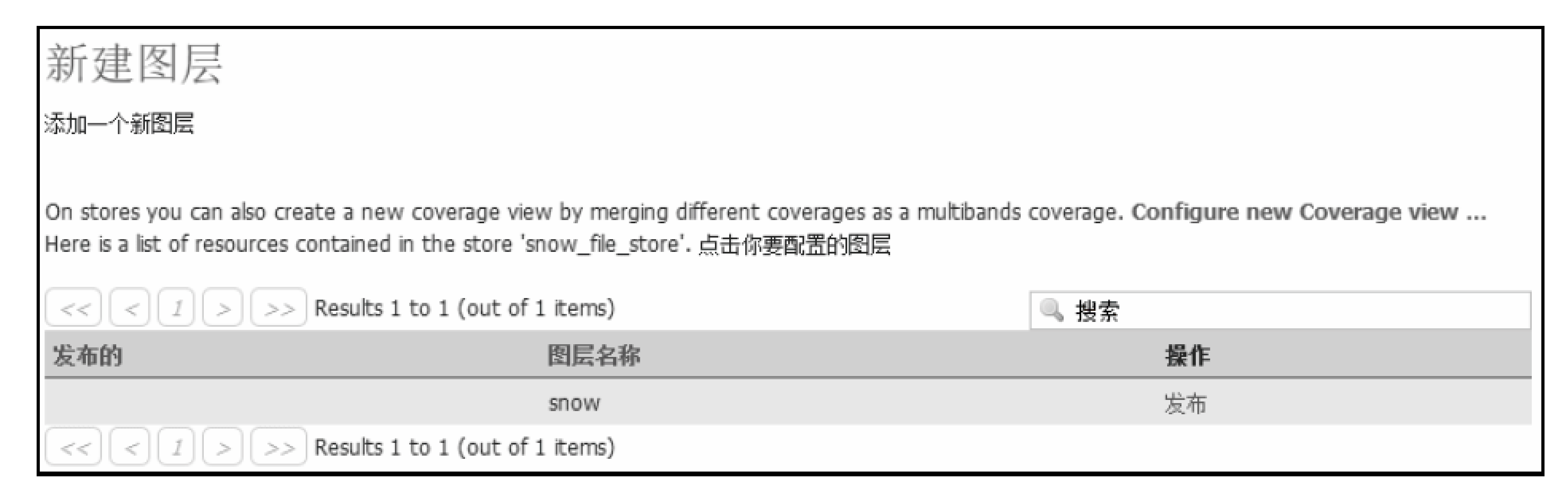


图 10.7　新建图层页面

在该页面中，单击“发布”按钮，进入“编辑图层”页面。

（6）配置图层及其发布信息。

在“编辑图层”页面中，基本配置信息都已经自动填好，需要设置的是“覆盖参数”部分。其中主要相关参数是 AllowMultithreading 与 USE_JAI_IMAGEREAD。并根据 Tiff 文件设置背景值。当同时又几个图像满足时间条件时，如果希望控制显示其中哪一个，那么则需要设置 SORTING 参数。在该参数中，首先要设置排序使用的变量名称，然后是一个空格，最后是 D 或 A。D 表示降序，A 表示升序。请按照图 10.8 设置参数。

覆盖参数

Accurate resolution computation

false

AllowMultithreading

false

BackgroundValues

Filter

FootprintBehavior

None

InputTransparentColor

MaxAllowedTiles

-1

MergeBehavior

FLAT

OutputTransparentColor

SORTING

ingestion D

SUGGESTED_TILE_SIZE

512, 512

USE_JAI_IMAGEREAD

true

图 10.8 设置覆盖参数

（7）设置样式。

为了能正确显示数据，需要使用特殊的样式。按照前面内容介绍的方法，将 snow 文件夹中的 snow_style.sld 上传到 GeoServer 中，创建名为 snow_style 的样式。并将 webgis:snow 图层的默认样式设置为 snow_style。

（8）设置时间属性。

在“编辑图层”页面中，切换到“维度”选项卡。首先在“时间”下面选择“启用”，在“简报”中选择“列表”，表示在 GetCapabilites 请求响应中包含列出所有可获取的时间，具体如下：

```
<Dimension name="time" default="2009-10-01T00:00:00Z" units="ISO8601">
  2009-10-01T00:00:00.000Z,2009-11-01T00:00:00.000Z,2009-12-01T00:00:00.000Z
</Dimension>
```

其他选项还有“连续间隔”与“间隔与分辨率”。时间属性的设置如图 10.9 所示。



图 10.9 时间属性设置

切换到“C:\Data\snowLZWdataset”文件夹中，打开 GeoServer 生成的 snowLZWdataset.properties 文件，其主要内容如下：

```
Levels=100.0,100.0
Heterogeneous=false
ElevationAttribute=elevation
TimeAttribute=ingestion
AbsolutePath=false
Name=snowLZWdataset
TypeName=snowLZWdataset
Caching=false
ExpandToRGB=false
LocationAttribute=location
SuggestedSPI=it.geosolutions.imageioimpl.plugins.tiff.TIFFImageReaderSpi
CheckAuxiliaryMetadata=false
LevelsNum=1
```

（9）图层预览。

对于时间序列影像，为了显示某个时间点的地图，需要在请求中增加时间参数，形式为“&time=<时间>”。

对于本实践，可先使用 OpenLayer 查看 webgis: snowLZWdataset 图层，也可以直接在浏览器中输入如下地址：

http://localhost:8080/geoserver/webgis/wms?service=WMS&version=1.1.0&request=GetMap&layers=webgis:snowLZWdataset&styles=&bbox=624800.0,5171500.0,632600.0,5184500.0&width=307&height=512&srs=EPSG:32632&format=application/openlayers

为了获得 2009 年 10 月、11 月与 12 月的地图，则可在上述地址后面分别加上&time=2009-10-01、&time=2009-11-01 与&time=2009-12-01 即可。

由于在发布服务时，将时间的默认值设置为了“Use the smallest domain value”，即最小值，因此在不带时间参数时默认显示的是 2009 年 10 月的数据。

通过对比 3 个图像，可以分析雪覆盖随时间的变化，如图 10.11 所示。颜色越深表示雪越多。

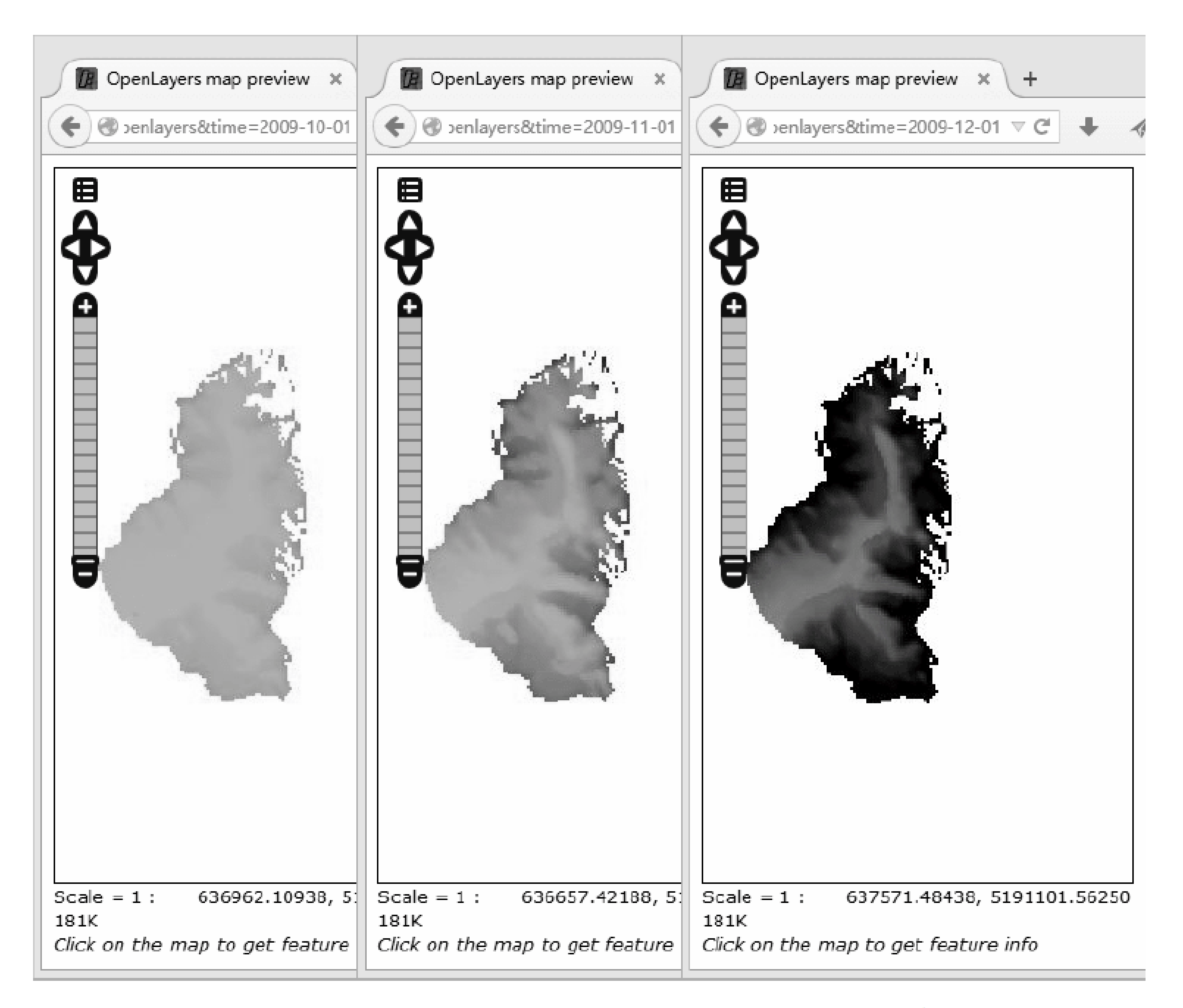


图 10.10　通过时间参数获取指定时间点的图像

（10）用 WCS 请求数据。

在前一步中，使用的是 OpenLayer 预览图层服务，其内部使用的是 WMS 服务。当然也可以使用 WCS 来请求原始数据。

首先使用 DescribeCoverage 获取覆盖描述。请求地址如下：

http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&CoverageId=webgis__snowLZWdataset&request=DescribeCoverage

该请求会显示创建的多维覆盖的详细信息。注意观察文档中有关时间维度的描述。相关内容如下：

```
<wcsgs:TimeDomain default="2009-12-01T00:00:00.000Z">
  <gml:TimeInstant gml:id="webgis__snowLZWdataset_td_0">
    <gml:timePosition>2009-10-01T00:00:00.000Z</gml:timePosition>
  </gml:TimeInstant>
  <gml:TimeInstant gml:id="webgis__snowLZWdataset_td_1">
    <gml:timePosition>2009-11-01T00:00:00.000Z</gml:timePosition>
```

```
    </gml:TimeInstant>
    <gml:TimeInstant gml:id="webgis__snowLZWdataset_td_2">
      <gml:timePosition>2009-12-01T00:00:00.000Z</gml:timePosition>
    </gml:TimeInstant>
</wcsgs:TimeDomain>
```

要使用 GetCoverage 操作得到某个时间点的覆盖数据，则需要加入一个时间的 subset 参数。例如要获取 2009 年 11 月 1 日的数据，可使用如下地址：

http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&CoverageId=webgis__snowLZWdataset&request=GetCoverage&format=geotiff&subset=http://www.opengis.net/def/axis/OGC/0/time("2009-11-01T00:00:00.000Z")

浏览器会下载一个名为 webgis__snowLZWdataset.tif 的文件，该文件中包含了请求的数据。

当然，我们不一定要指定原始文件中的几个时间点，例如我们可以使用如下 URL 请求 2009 年 11 月 15 日的数据：

http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&CoverageId=webgis__snowLZWdataset&request=GetCoverage&format=geotiff&subset=http://www.opengis.net/def/axis/OGC/0/time("2009-11-15T00:00:00.000Z")

10.3.2 发布时间序列与高程序列栅格数据

下面将介绍如何发布同时带有时间序列与高程系列的栅格数据。

（1）数据复制。

建立一个简单的文件夹，例如 “C:\Data\ temperatureLZWdataset”。并将下载文件“Data\ temperature”文件夹中的 4 个 Tiff 文件复制进来。这几个文件的命名规则是：

```
{覆盖名称}_{时间戳}_{高程}.tiff
```

（2）设置配置文件。

对于同时带有时间序列与高程系列的栅格数据，需要比只带有时间需要的数据多增加一个 elevationregex.properties 文件，用于规定如何从文件名中提取高程信息。

在“C:\Data\temperatureLZWdataset”文件夹中创建一个名为 indexer.properties 的文本文件，其内容如下：

```
Caching=false
TimeAttribute=ingestion
ElevationAttribute=elevation
Schema=*the_geom:Polygon,location:String,ingestion:java.util.Date,elevation:Double
PropertyCollectors=TimestampFileNameExtractorSPI[timeregex](ingestion),DoubleFileNameExtractorSPI[elevationregex](elevation)
```

在“C:\Data\temperatureLZWdataset”文件夹中创建一个名为 timeregex.properties 的文本文件，其内容如下：

```
regex=[0-9]{8}T[0-9]{9}Z(\?!.\*[0-9]{8}T[0-9]{9}Z.\*)
```

在“C:\Data\temperatureLZWdataset”文件夹中创建一个名为 elevationregex.properties 的文本文件，其内容如下：

```
regex=(?<=_)(\\d{4}\\.\\d{3})(?=_)
```

（3）创建图像镶嵌数据存储。

在 GeoServer 的 Web 管理页面中，新建一个 ImageMosaic 类型的数据存储。将工作区选择为 webgis，数据源名称设置为“temperature_file_store”，URL 设置为在第一步中创建的目录，即 C:\Data\temperatureLZWdataset。

（4）发布图层。

发布图层与前面类似，只是需要再启用高程。设置如图 10.11 所示。

webgis:temperatureLZWdataset
配置当前图层的和发布信息
数据　发布　维度　Tile Caching
时间
☑ 启用
简报
列表
Default value
Use the smallest domain value
高程
☑ 启用
单位
EPSG:5030
单位符号
m
简报
列表
Default value
Use the smallest domain value

图 10.11　同时启用时间与高程维度

（5）图层阅览。

为了显示某个时间点的某高程栅格数据的地图，需要在请求中增加时间与高程参数，形式为“&time= <时间>&elevation=<高程>”。如果没有这些参数将使用默认时间点与高程。

例如对于 2013 年 3 月 10 日下午 6 点在海拔 200 米处的气温覆盖地图，可使用如下地址来获得：

```
http://localhost:8080/geoserver/webgis/wms?service=WMS&version=1.1.0&request=GetMap&layers=webgis:temperatureLZWdataset&styles=&bbox=-30.0,25.0,45.0,70.0&width=550&height=330&srs=EPSG:4326&format=application/openlayers&time=2013-03-10T18:00:00.000Z&elevation=200.0
```

返回的图像如图 10.12 所示。

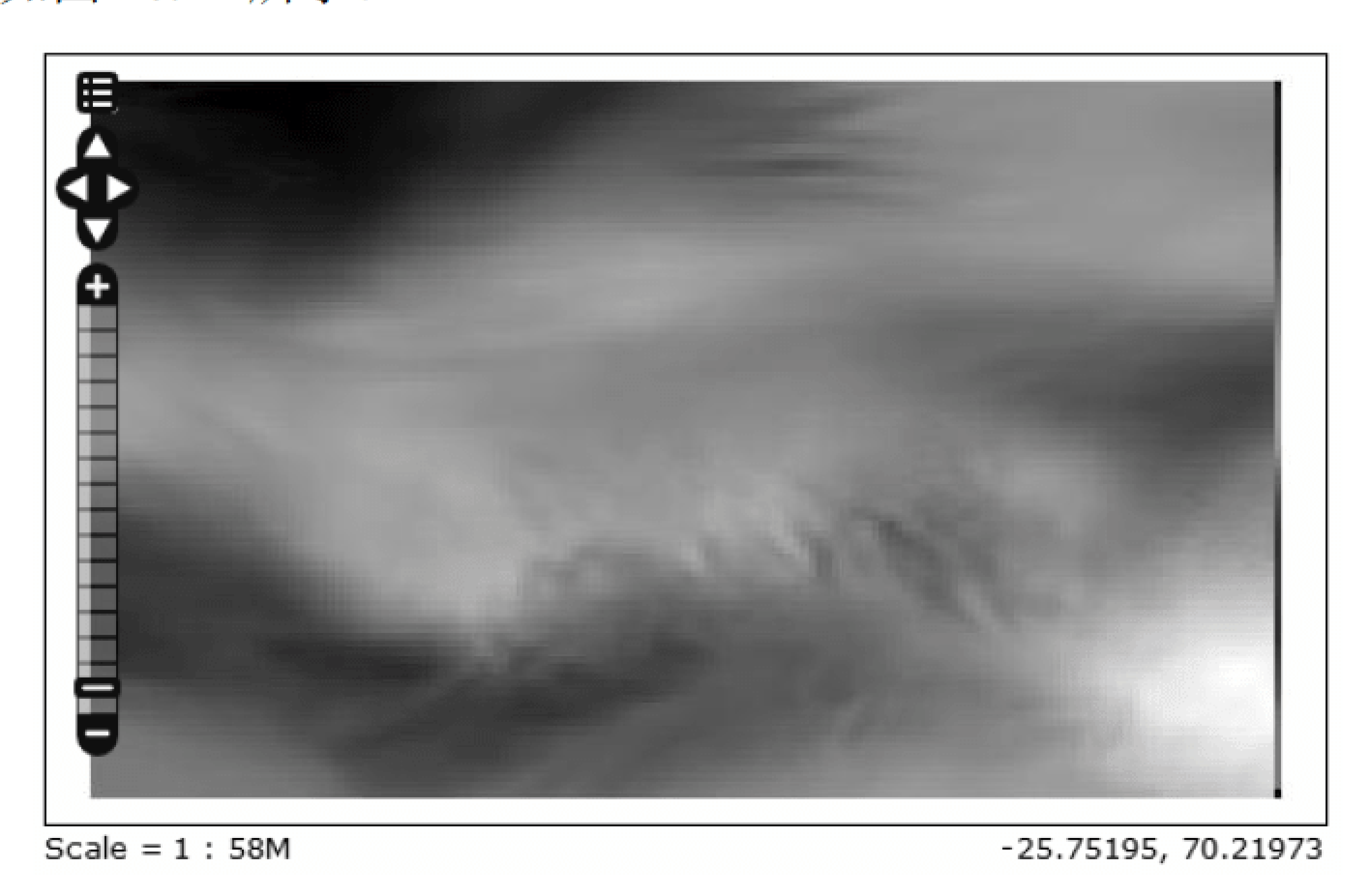


图 10.12　某个时间点的某高程处的气温覆盖

而对于同一个时间点的 300 米高程处气温覆盖，其获取请求地址最后部分为&time=2013-03-10T18:00:00.000Z&elevation=300.0。

（6）用 WCS 请求数据。

使用如下 DescribeCoverage 请求获取覆盖的详细信息：

http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&CoverageId=webgis__temperatureLZWdataset&request=DescribeCoverage

时间维度内容如下，包含两个时间点：

```
<wcsgs:TimeDomain default="2013-03-11T18:00:00.000Z">
  <gml:TimeInstant gml:id="webgis  temperatureLZWdataset td 0">
    <gml:timePosition>2013-03-10T18:00:00.000Z</gml:timePosition>
  </gml:TimeInstant>
  <gml:TimeInstant gml:id="webgis__temperatureLZWdataset_td_1">
    <gml:timePosition>2013-03-11T18:00:00.000Z</gml:timePosition>
  </gml:TimeInstant>
</wcsgs:TimeDomain>
```

高程维度内容如下，包含两个高程项：

```
<wcsgs:ElevationDomain default="200.0" uom="m">
  <wcsgs:SingleValue>200.0</wcsgs:SingleValue>
  <wcsgs:SingleValue>300.0</wcsgs:SingleValue>
</wcsgs:ElevationDomain>
```

可使用如下请求地址来获取 2013 年 3 月 11 日下午 6 点海拔 300 米的气温覆盖数据：

```
http://localhost:8080/geoserver/ows?service=WCS&version=2.0.1&CoverageId=webgis__temperatureLZWdataset&request=GetCoverage&format=geotiff&subset=http://www.opengis.net/def/axis/OGC/0/time("2013-03-11T18:00:00.000Z")&subset=http://www.opengis.net/def/axis/OGC/0/elevation(300)
```

10.4　实践 16：在 OpenLayers 中访问 WCS

在 OpenLayers 中，并没有专门用于访问 WCS 服务的类，对于简单 WCS 服务可以使用 OpenLayers.Layer.Image 类来实现。

10.4.1　页面设计

新建一个名为 Walkthrough16 的文件夹，在其中新建一个名为 GetCoverage_v2_0_1.html 的文件。

在 GetCoverage_v2_0_1.html 页面中加入如下代码：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>OpenLayers WCS GetCoverage 实例</title>
    <script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
    </script>
    <script></script>
</head>
<body onload="init()">
    <h1 id="title">WCS GetCoverage (v2.0.1)实例</h1>
    <div id="map" style="width:500px; height:500px;">
     </div>
    <br/>
    <div id="docs">
        可使用覆盖：
        <select id="CoverageIDget" onchange="setCoverageExtent();">
            <option value="-180.0,-90.0,180.0,90.0">nurc__Arc_Sample</option>
            <option value="-130.85168,20.7052,-62.0054,54.1141"> nurc   Img  Sample
            </option>
            <option value="6.346,36.492,20.83,46.591">nurc__mosaic</option>
        </select>
```

```
            <button id="getCoverageBtn" type="button" onclick="getCoverage();">
                <img id="getCoverageImg" class="btnImg" src="arrow.png"/>  
                GetCoverage
            </button>
        </div>
</body>
</html>
```

10.4.2 代码设计

（1）初始化地图。

在<script>与</script>之间加入如下代码：

```
var map, WCSimg;
var fullExtLayer;
var xmaxFullExt, ymaxFullExt, xminFullExt, yminFullExt;

function init() {
    map = new OpenLayers.Map('map');
    // 设置基础底图
    var ol_wms = new OpenLayers.Layer.WMS(
        "OpenLayers WMS",
        "http://vmap0.tiles.osgeo.org/wms/vmap0",
        {
            layers: "basic",
            maxExtent: [-180, -88.759, 180, 88.759],
            isBaseLayer: true
        }
    );
    map.addLayer(ol_wms);

    // 显示鼠标位置处的坐标
    map.addControl(new OpenLayers.Control.MousePosition());
    map.addControl(new OpenLayers.Control.LayerSwitcher({ 'ascending': false }));

    // 用于绘制覆盖范围的矢量图层
    var fullExtLayer_style = OpenLayers.Util.extend({}, OpenLayers.Feature.Vector.style['default']);
    fullExtLayer_style.strokeColor = "#FF0000";
    fullExtLayer_style.strokeWidth = 2;
    fullExtLayer_style.strokeDashstyle = "dashdot";
```

```
        fullExtLayer = new OpenLayers.Layer.Vector("AOI  Polygons", {
            displayInLayerSwitcher: false,
            isBaseLayer: false,
            style: fullExtLayer_style
        });

        map.addLayer(fullExtLayer);

        // 设置当前选择的覆盖的范围与图像
        setCoverageExtent();

        // 为了确保矢量图层位于顶部，创建了一个假的活动控件
        var fullExtOnTopControl = new OpenLayers.Control.SelectFeature(fullExtLayer, {
            hover: false,
            autoActivate: true
        });
        map.addControl(fullExtOnTopControl);
    }
```

（2）设置选择的覆盖的空间范围。

setCoverageExtent 函数主要用于获取当前选择的覆盖的空间范围，然后在 fullExtLayer 矢量图层中绘制表示该范围的图形，最后调用 getCoverage 函数获取覆盖的图像。代码如下：

```
    function setCoverageExtent() {
        var e = document.getElementById("CoverageIDget");
        var c_bbox = e.options[e.selectedIndex].value;
        var c  bboxList = c  bbox.split(",");

        xminFullExt = Number(c_bboxList[0]);
        yminFullExt = Number(c_bboxList[1]);
        xmaxFullExt = Number(c  bboxList[2]);
        ymaxFullExt = Number(c  bboxList[3]);

        var fullExtLineGeom = new OpenLayers.Geometry.LineString([
            new OpenLayers.Geometry.Point(xminFullExt, yminFullExt),
            new OpenLayers.Geometry.Point(xmaxFullExt, yminFullExt),
            new OpenLayers.Geometry.Point(xmaxFullExt, ymaxFullExt),
            new OpenLayers.Geometry.Point(xminFullExt, ymaxFullExt),
            new OpenLayers.Geometry.Point(xminFullExt, yminFullExt)
        ]);

        fullExtLayer.destroyFeatures();
```

```
        fullExtLayer.addFeatures([new OpenLayers.Feature.Vector(fullExtLineGeom)]);
        fullExtLayer.redraw();

        // 将地图缩放到覆盖的空间范围
        map.zoomToExtent(new OpenLayers.Bounds(xminFullExt, yminFullExt, xmaxFullExt, ymaxFullExt));

        // 将覆盖作为一个图像获取
        getCoverage();
    }
```

（3）利用 WCS 的 GetCoverage 操作获取覆盖的图像。

getCoverage 函数用于获取覆盖的图像，其代码如下：

```
    function getCoverage() {
        var imgUrl, imgBounds;

        var e = document.getElementById("CoverageIDget");
        var c_id = e.options[e.selectedIndex].text;

        imgUrl = 'http://localhost:8080/geoserver/wcs?service=WCS&version=2.0.1&CoverageId=' + c_id +
'&request=GetCoverage&format=image/png&';
        imgBounds = new OpenLayers.Bounds(xminFullExt, yminFullExt, xmaxFullExt, ymaxFullExt);

        if (WCSimg) {
            map.removeLayer(WCSimg);
        }

        // 用于 WCS 的图像图层
        WCSimg = new OpenLayers.Layer.Image(
            'GetCoverage PNG',
            imgUrl,
            imgBounds,
            new OpenLayers.Size(10, 10),
            { isBaseLayer: false}
        );

        WCSimg.events.on({
            loadstart: function () {
                document.getElementById("getCoverageBtn").disabled = true;
                document.getElementById("getCoverageImg").src = "ajax-loader.gif";
            },
            loadend: function () {
```

```
                document.getElementById("getCoverageBtn").disabled = false;
                document.getElementById("getCoverageImg").src = "arrow.png";
            }
        });

        map.addLayer(WCSimg);

        // 将地图缩放到覆盖的空间范围
        map.zoomToExtent(imgBounds);
    }
```

（4）程序运行与调试。

用 Firefox 等浏览器直接打开 GetCoverage_v2_0_1.html 文件，然后通过下拉框选择需要显示的覆盖，如图 10.13 所示。如果程序运行有错误，请参看下载文件“Codes\Walkthrough15”文件夹中的 GetCoverage_v2_0_1.html 文件。



图 10.13　程序运行效果

10.5　习　题

（1）在互联网上寻找带有时间序列与高程序列的多维栅格数据，使用 GeoServer 的图像镶嵌插件将其发布为多维 WCS 服务。

（2）研究如何在 GeoServer 中创建覆盖视图。

第 11 章

Web 处理服务

从本章可以学习到：

- GeoServer 中的 WPS
- WPS 的操作
- 使用 WPS 创建等高线地图
- 在 OpenLayers 中使用 WPS

前面介绍的 WMS、WFS 提供了地图绘制与空间数据服务，在地理信息系统中，另一个重要的功能是地理处理，包括缓冲区分析、空间变换、网络分析以及空间插值等。开源 GIS 或商业 GIS 软件都包含有地理处理功能，在前面的内容中介绍了使用 QGIS 与 GDAL 执行空间坐标系变换等功能。但是如何让用户从 Web 浏览器中运行这些功能呢？例如希望提供如下功能：用户在 Web 地图上绘制一个多边形，然后计算该多边形内的人口和诊所的数量。这样不需要每个用户都安装一套 GIS 软件，只需要在服务器上进行空间数据处理，便可将该强大而实用的空间分析功能提供给广大用户。

同样的，这类地理处理功能也都可以通过 Web 服务的形式对外提供使用。而 OGC 的 Web 处理服务（WPS）规范就是一种用于在 Web 上提供和执行这类地理空间处理的国际规范，用于在不同的平台和客户端之间以一种开放并经过认可的方式提供地理处理服务。

本章将介绍 WPS 及其服务的发布、访问与应用，并通过实践的方式介绍如何通过该服务实现基于 Web 的等高线生成以及空间数据的处理。

11.1　GeoServer 中的 WPS

GeoServer 默认并不提供 WPS 服务，需要单独安装 WPS 扩展。而且该扩展包含两大方面的内容，一方面是开源的 Java 拓扑套件（Java Topology Suite，简写为 JTS，网址为 http://tsusiatsoftware.net/jts/main.html）提供的地理处理，另一方面是 GeoServer 开发的其他地理处理。

11.1.1　WPS 扩展的安装

在 GeoServer 的下载页面（http://geoserver.org/download/）中找到对应版本的 GeoServer（本书使用的是 2.6.2），然后下载 WPS 扩展。也可以直接使用下载文件“Resources\Tools”文件夹中的 geoserver-2.6.2-wps-plugin.zip 文件。

将 ZIP 文件解压，并加其中所有的文件复制到 GeoServer 安装路径中的“WEB-INF/lib”文件夹中，如果是默认安装的话，目录如下：

C:\Program Files\GeoServer 2.6.2\webapps\geoserver\WEB-INF\lib

重新启动 GeoServer，便可在 GeoServer 的 Web 管理页面的服务能力中看到新加入的 WPS，如图 11.1 所示。如果没有出现，说明安装未成功。

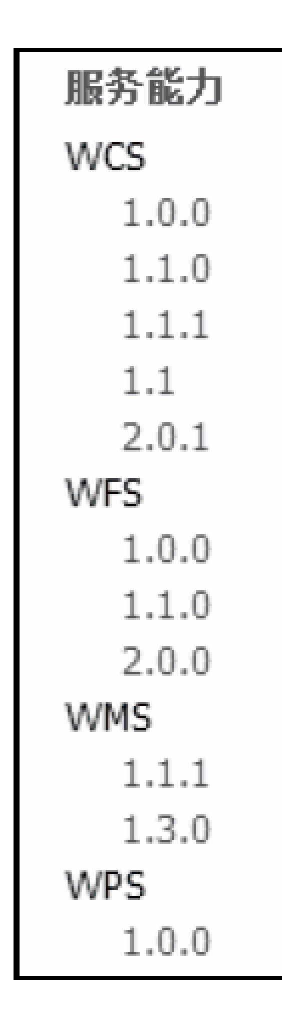


图 11.1　成功安装 WPS 扩展后 GeoServer 的服务能力列表

11.1.2　GeoServer 中 WPS 包含的类型

GeoServer 实现了两大类的地理处理，一类是 Java 拓扑套件，另一类是 GeoServer 自身开发的。

（1）Java 拓扑套件。

Java 拓扑套件是一套用于处理几何要素拓扑关系的函数库。它提供了完整、稳定、可靠的基本二位平面线形图形运算算法实现，包括面积计算、缓冲区分析、相交计算与图形简化等。这类地理处理的 ID 以 JTS 开头，例如 JTS:area、JTS:boundary 等。

（2）GeoServer 自身的地理处理。

这类地理处理是 GeoServer 特殊开发的，一般只能用于 GeoServer 中，例如边界计算与投影变换等。主要用途是在 GeoServer 内部连接 WFS/WCS 服务，用于读写数据，并不是 WPS 规范的一部分。这类地理处理的 ID 以 geo、gs、ras 与 vec 开头，例如 geo:polygonize、gs:contour 等。

虽然 GeoServer 提供的地理过程有限，但是 WPS 的一个很大的优点是可将这些地理过程连接起来，就像在函数中调用其他函数一样。对于 WPS 一个地理处理可以使用其他地理处理的输出作为输入，因此可将多个地理处理组合起来，形成一个强大的功能。

例如，GeoServer 中自带了如下两个图层：

- sf:roads：包含道路信息的图层；
- sf:restricted：表示限制区域的图层。

如图 11.2 所示，底图是 DEM，道路用白线表示，限制区域是用虚线绘制的多边形，而限制区域内的道路用深色表示。现在要计算有多少公里的道路穿越了保护区域。

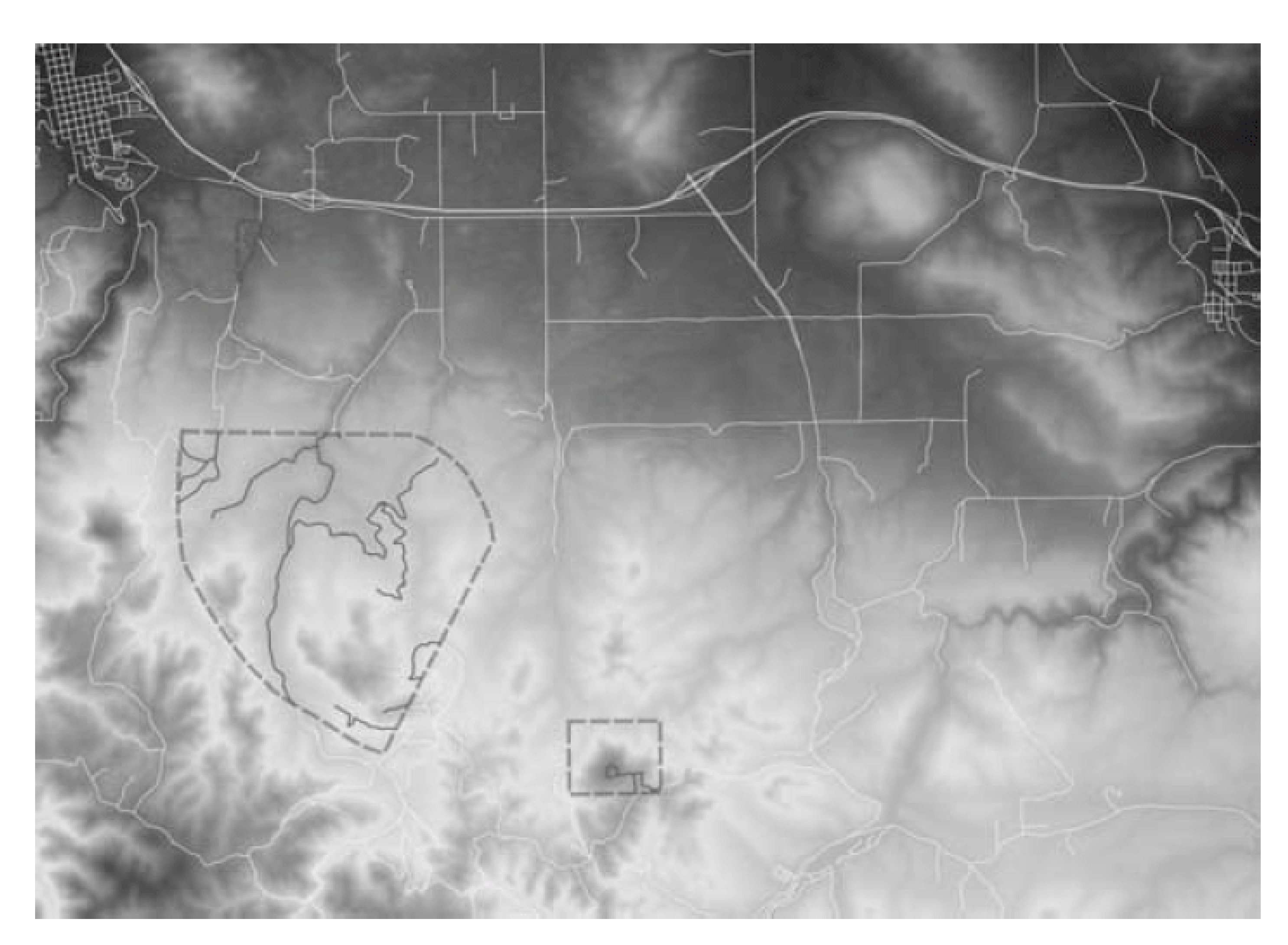


图 11.2　道路网与限制区域图

为了计算总长度，需要使用到 GeoServer 中内建的如下一些 WPS 地理处理：

- gs:IntersectionFeatureCollection，将道路要素集与限制区域要素集进行相交计算，返回相交要素集；
- gs:CollectGeometries，从相交要素集中获取所有的几何对象，返回几何对象集合；
- JTS:length，计算所有几何对象的长度和。

11.2　WPS 的操作

在开源领域中，除了 GeoServer 之外，ZOO 开放 WPS 平台（http://www.zoo-project.org/）与 PyWPS（http://pywps.wald.intevation.org/）也可提供了 WPS 服务。在商业软件领域，ESRI 的 ArcGIS Server 可将 ModelBuilder 中的模型发布为 WPS 服务。

像其他 OGC 服务一样，WPS 也提供了一组可调用的操作，分别是 GetCapabilities、DescribeProcess 与 Execute。

11.2.1　GetCapabilities 操作

GetCapabilities 请求返回服务的基本元数据。对于本地安装的 GeoServer，可使用如下请求地址获取其 WPS 服务的元数据：

http://localhost:8080/geoserver/ows?service=wps&version=1.0.0&request=GetCapabilities

从返回的响应可以看出，主要包含两部分内容：

（1）该 WPS 服务支持的操作。

（2）WPS 服务中提供的地理处理列表，以及每个地理处理的简单描述。例如：

```
    <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>JTS:area</ows:Identifier>
      <ows:Title>Area</ows:Title>
      <ows:Abstract>Returns the area of a geometry, in the units of the geometry. Assumes a Cartesian plane, so this process is only recommended for non-geographic CRSes.</ows:Abstract>
    </wps:Process>
    <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>JTS:boundary</ows:Identifier>
      <ows:Title>Boundary</ows:Title>
      <ows:Abstract>Returns a geometry boundary. For polygons, returns a linear ring or multi-linestring equal to the boundary of the polygon(s). For linestrings, returns a multipoint equal to the endpoints of the linestring. For points, returns an empty geometry collection.</ows:Abstract>
    </wps:Process>
```

11.2.2 DescribeProcess 操作

GetCapabilities 操作得到了可执行的地理处理的 ID 及其简单描述，要得到某个地理处理的详细描述，则需要使用 DescribeProcess 操作。

如果对 JTS:buffer 感兴趣，则可构造如下请求，获取其详细信息：

http://localhost:8080/geoserver/ows?service=wps&version=1.0.0&request=DescribeProcess&Identifier=JTS:buffer

WPS 的地理处理描述文档中主要包含了执行该处理需要的输入参数与输出结果。

输出参数在 DataInputs 中。例如对于 JTS:buffer 地理处理，在其描述文档中指出了要求以下一些输入参数：

```
    <DataInputs>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>geom</ows:Identifier>
        <ows:Title>geom</ows:Title>
        <ows:Abstract>Input geometry</ows:Abstract>
        <ComplexData>
          <Default>
            <Format>
              <MimeType>text/xml; subtype=gml/3.1.1</MimeType>
            </Format>
          </Default>
        </ComplexData>
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>distance</ows:Identifier>
        <ows:Title>distance</ows:Title>
```

```
        <ows:Abstract>Distance to buffer the input geometry, in the units of the geometry</ows:Abstract>
        <LiteralData>
          <ows:DataType>xs:double</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
      <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>quadrantSegments</ows:Identifier>
        <ows:Title>quadrantSegments</ows:Title>
        <LiteralData>
          <ows:DataType>xs:int</ows:DataType>
          <ows:AnyValue/>
        </LiteralData>
      </Input>
      <Input minOccurs="0" maxOccurs="1">
        <ows:Identifier>capStyle</ows:Identifier>
        <ows:Title>capStyle</ows:Title>
        <LiteralData>
          <ows:AllowedValues>
            <ows:Value>Round</ows:Value>
            <ows:Value>Flat</ows:Value>
            <ows:Value>Square</ows:Value>
          </ows:AllowedValues>
        </LiteralData>
      </Input>
    </DataInputs>
```

从上述内容，可以看到 JTS:buffer 需要两个必选参数，分别是 geom（要进行缓冲区计算的几何对象）与 distance（缓冲距离），还可以包含两个可选参数，分别是 quadrantSegments（表示 1/4 圆所使用的线段数）与 capStyle（缓冲区末端的形状）。

输出信息由 ProcessOutputs 部分指定。

11.2.3　Execute 操作

Execute 操作用于请求运行一个由 WPS 服务实现的地理处理。请求可以是一个 GET 的 URL，或是带 XML 文档的 POST 请求。不过，由于请求包含比较复杂的结构，通常使用 POST 方式。

下面是缓冲区计算的 Execute 操作的 POST 请求实例。在该实例中，输出参数 geom 对应的是一点对象(POINT(0 0))，距离 distance 参数指定为 10，quadrantSegments 设置为 1，capStyle 设置为 flat。ResponseForm 元素指定了输出结果格式为 GML 3.1.1。

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute version="1.0.0" service="WPS" >
  <ows:Identifier>JTS:buffer</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>geom</ows:Identifier>
      <wps:Data>
        <wps:ComplexData mimeType="application/wkt">
          <![CDATA[POINT(0 0)]]>
        </wps:ComplexData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>10</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>quadrantSegments</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>1</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>capStyle</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>flat</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/gml-3.1.1">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

可以使用 Fiddler 等工具来发送 POST 请求。

11.3　实践 17：使用 WPS 创建等高线地图

该实践将演示如何从一个栅格 DEM 图层创建一个等高线地图，展示了两种不同的方法：

（1）创建一个静态的矢量等高线图层；

（2）利用着色器转换动态创建等高线样式，从而用等高线的方式显示高程数据。

11.3.1　创建静态等高线地图

为了方便，本实践直接使用 GeoServer 自带的 sf 工作区的 sfdem 图层，以作为原始 DEM 数据。

（1）查看原始数据。

登录进入 GeoServer 的 Web 管理页面，使用 OpenLayer 预览 sf:sfdem 图层，得到图 11.3 所示的栅格地图。

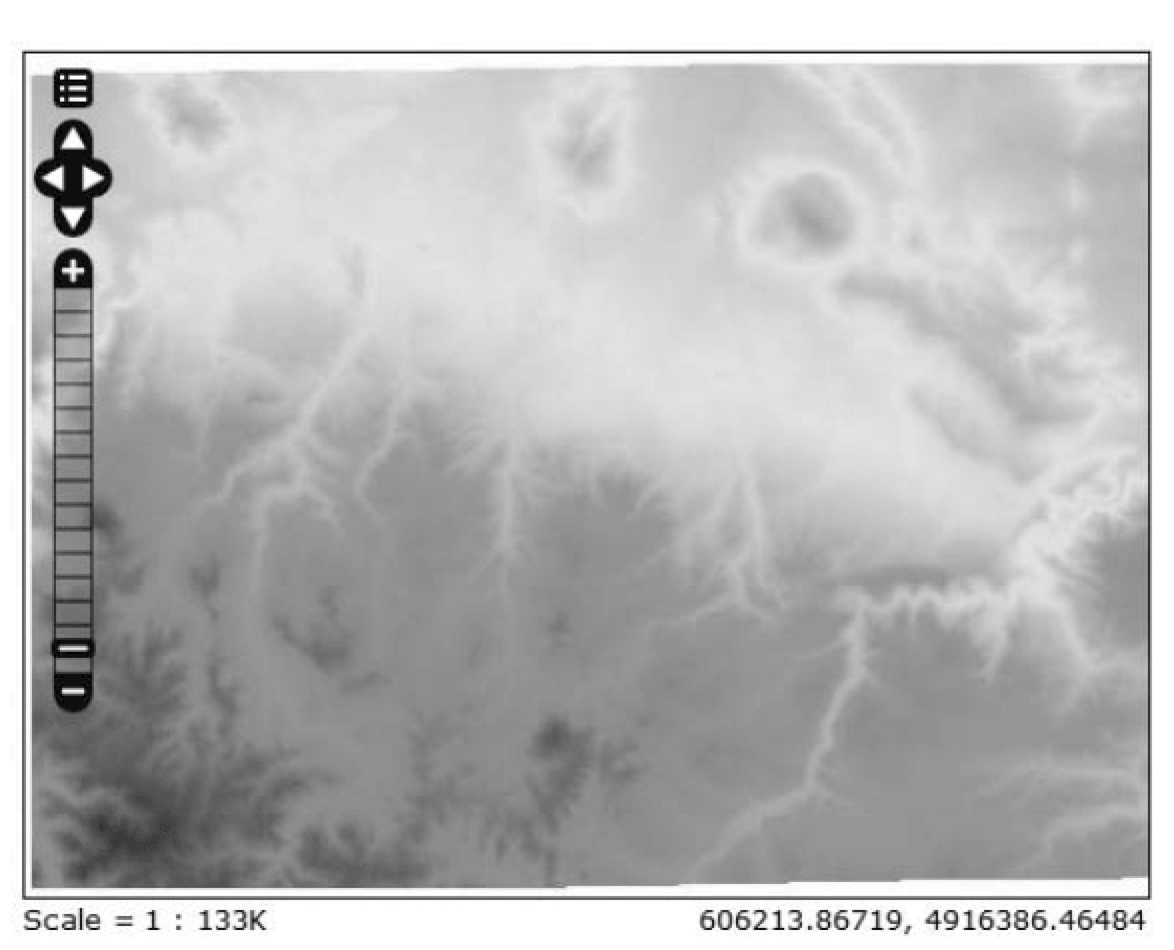


图 11.3　使用 OpenLayer 预览 sf 工作区的 sfdem 图层

（2）进入 WPS 请求构造页面。

在 GeoServer 的 Web 管理页面的左边菜单中选择“演示”，进入图 11.4 所示的演示程序列表页面，在其中选择最下面的“WPS request builder”，进入 WPS 请求构造页面。

GeoServer的演示程序

GeoServer的演示程序集合

- Demo requests Example requests for GeoServer (using the TestServlet).
- SRS List List of all SRS known to GeoServer
- Reprojection console Simple coordinate reprojection tool
- WCS request builder Step by step WCS GetCoverage request builder
- WPS request builder Step by step WPS request builder

图 11.4　GeoServer 演示程序列表

（3）构造 WPS 请求。

在 WPS 请求构造页面中，首先选择“gs:Contour”作为地理处理程序，然后在输入栅格图层中选择 sf:sfdem 作为输入图层，将创建等高线所使用值设置为 1200，在是否需要简化文本框中输入 true，设置如图 11.5 所示。

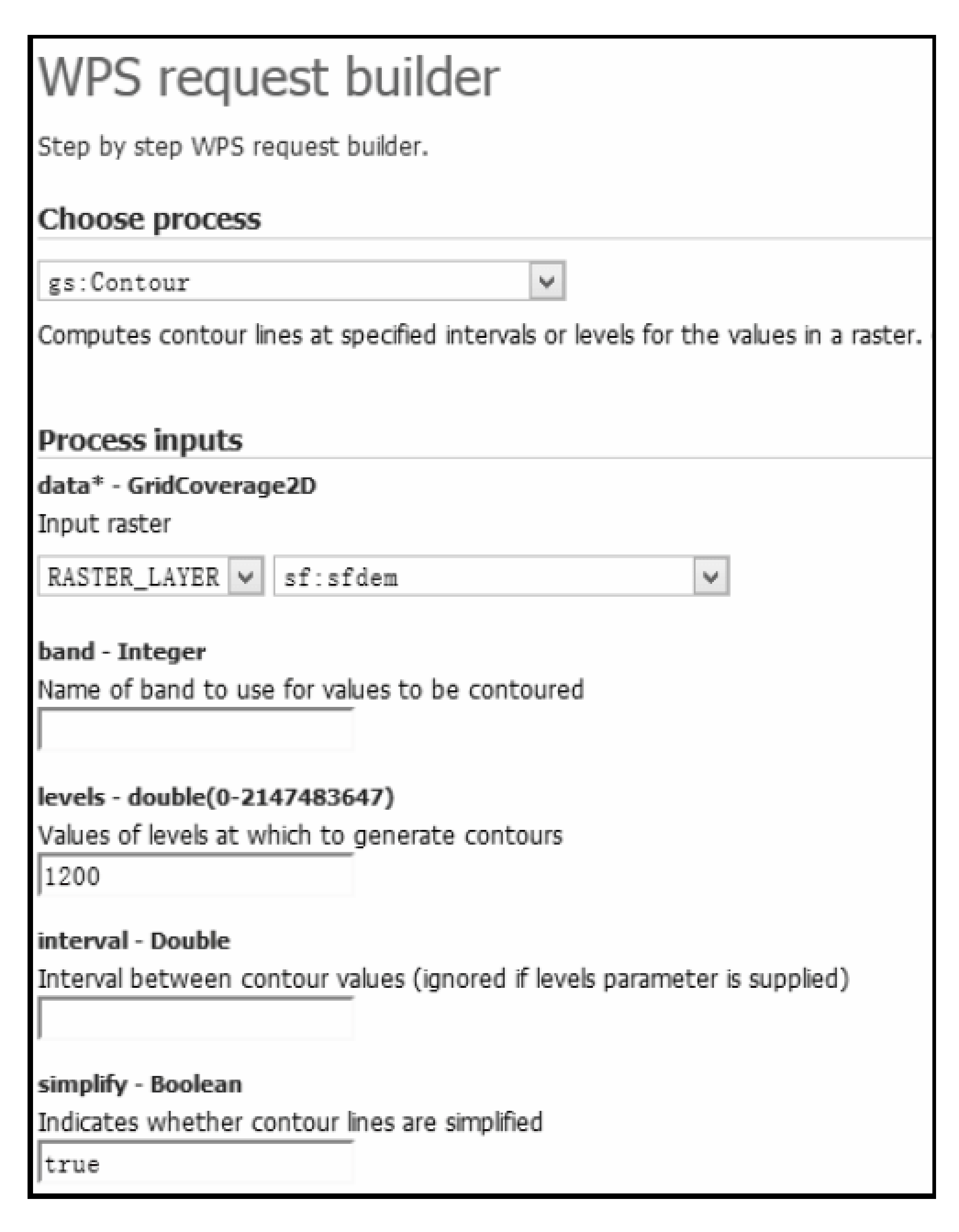


图 11.5　构造创建等高线地理处理过程请求

最后在该页面的输出结果部分设置为“application/zip”，如图 11.6。

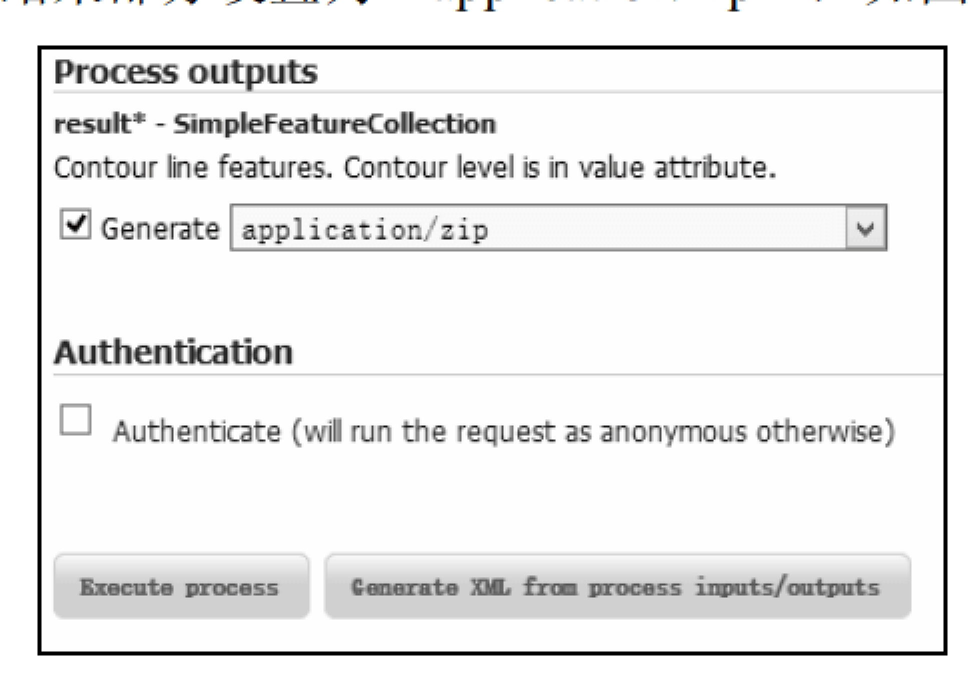


图 11.6　设置输出格式

（4）生成请求 XML 文档。

在 WPS 请求构造页面的最底部，单击“Generate XML from process inputs/outputs”，将得到如下的 XML 文档：

```
<?xml version="1.0" encoding="UTF-8"?><wps:Execute version="1.0.0" service="WPS" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" >
```

```
    <ows:Identifier>gs:Contour</ows:Identifier>
    <wps:DataInputs>
      <wps:Input>
        <ows:Identifier>data</ows:Identifier>
        <wps:Reference mimeType="image/tiff" xlink:href="http://geoserver/wcs" method="POST">
          <wps:Body>
            <wcs:GetCoverage service="WCS" version="1.1.1">
              <ows:Identifier>sf:sfdem</ows:Identifier>
              <wcs:DomainSubset>
                <gml:BoundingBox crs="http://www.opengis.net/gml/srs/epsg.xml#26713">
                  <ows:LowerCorner>589980.0 4913700.0</ows:LowerCorner>
                  <ows:UpperCorner>609000.0 4928010.0</ows:UpperCorner>
                </gml:BoundingBox>
              </wcs:DomainSubset>
              <wcs:Output format="image/tiff"/>
            </wcs:GetCoverage>
          </wps:Body>
        </wps:Reference>
      </wps:Input>
      <wps:Input>
        <ows:Identifier>levels</ows:Identifier>
        <wps:Data>
          <wps:LiteralData>1200</wps:LiteralData>
        </wps:Data>
      </wps:Input>
      <wps:Input>
        <ows:Identifier>simplify</ows:Identifier>
        <wps:Data>
          <wps:LiteralData>true</wps:LiteralData>
        </wps:Data>
      </wps:Input>
    </wps:DataInputs>
    <wps:ResponseForm>
      <wps:RawDataOutput mimeType="application/zip">
        <ows:Identifier>result</ows:Identifier>
      </wps:RawDataOutput>
    </wps:ResponseForm>
  </wps:Execute>
```

（5）修改请求 XML 文档。

由于我们需要的是高程每隔 100 米就创建一条等高线，然而在 WPS 请求构造器中设置参数时，并不能自动根据输入的多个值而生成多个输入参数，因此需要我们手工加入。

在上述的 XML 文档的<wps:DataInputs>与</wps:DataInputs>之间，加入其他高程值作为输入参数，加入内容如下：

```
<wps:Input>
  <ows:Identifier>levels</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>1300</wps:LiteralData>
  </wps:Data>
</wps:Input>
<wps:Input>
  <ows:Identifier>levels</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>1400</wps:LiteralData>
  </wps:Data>
</wps:Input>
<wps:Input>
  <ows:Identifier>levels</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>1500</wps:LiteralData>
  </wps:Data>
</wps:Input>
<wps:Input>
  <ows:Identifier>levels</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>1600</wps:LiteralData>
  </wps:Data>
</wps:Input>
<wps:Input>
  <ows:Identifier>levels</ows:Identifier>
  <wps:Data>
    <wps:LiteralData>1700</wps:LiteralData>
  </wps:Data>
</wps:Input>
```

这就是需要发送到服务器的 POST 请求的内容。

（6）Fiddler 工具安装。

为了将上述 POST 请求发送到 GeoServer 服务器，执行地理处理操作，需要使用其他工具。本书选择使用 Fiddler。从下载文件的“Resources\Tools”文件夹中复制 fiddler-4-4-9-6-en-win.exe 文件，然后双击安装 Fiddler。

（7）利用 Fiddler 工具发送 POST 请求。

运行 Fiddler，在其窗口的右上部单击“Composer”按钮，进入 HTTP 请求构造器选项卡。

在该选项卡中，将请求方式设置为 POST，请求地址设置为 http://localhost:8080/geoserver/wps，并将上述修改后的 XML 请求文档内容复制到 Request Body 中，如图 11.7 所示。最后单击“Execute”按钮，向服务器发送请求。

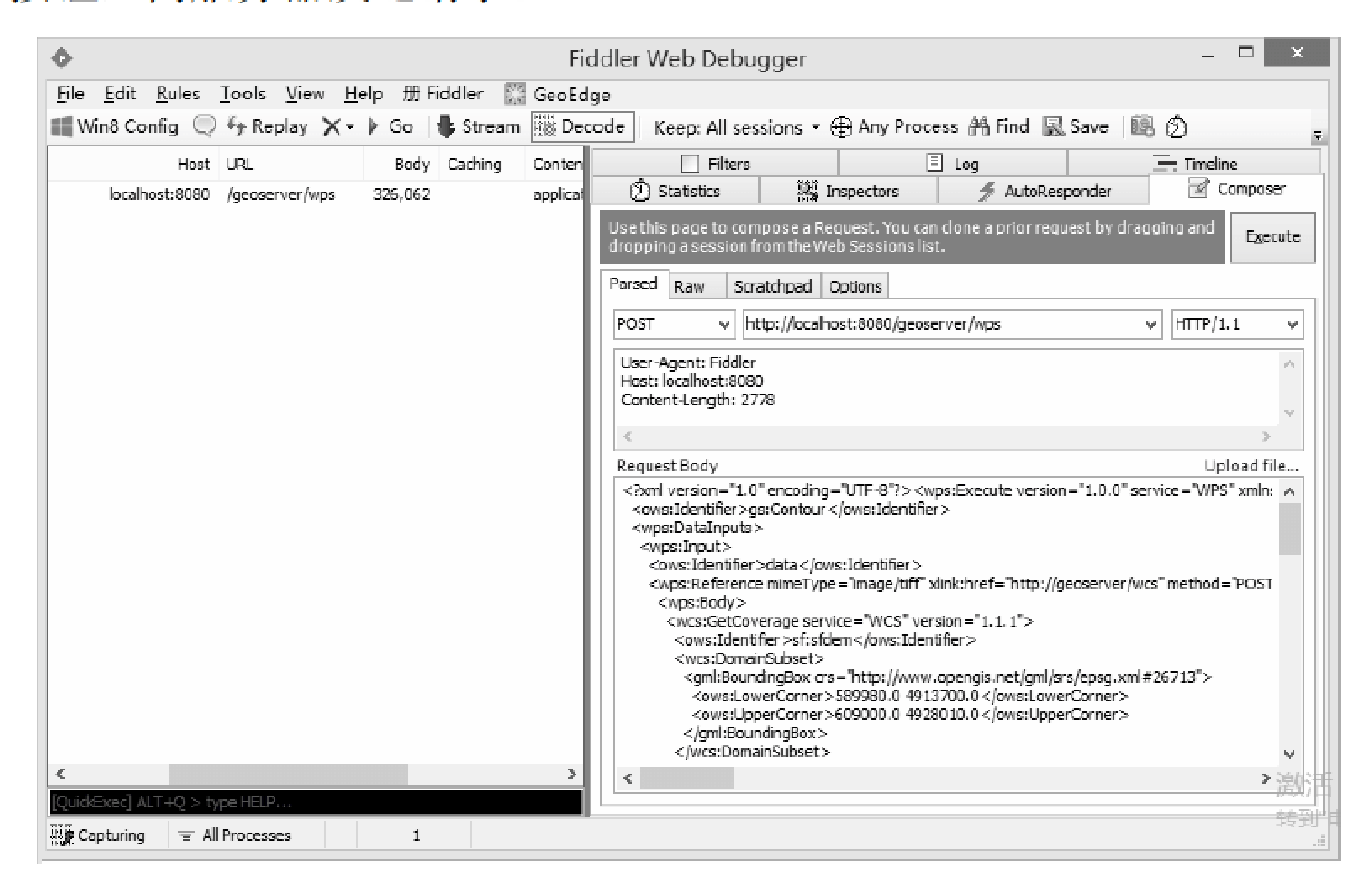


图 11.7　使用 Fiddler 工具发送 POST 请求

（8）获取返回结果。

发送请求之后，在 Fiddler 窗口的左边就会列出该请求。高亮选择该请求，然后选择 File 菜单中的 Save->Response->Response Body，将请求响应结果保存为 result.zip。该压缩文件中包含矢量等高线 sfdem 的 Shapefile 文件。

可使用 QGIS 打开该 Shapefile 文件，得到图 11.8 所示的等高线地图。

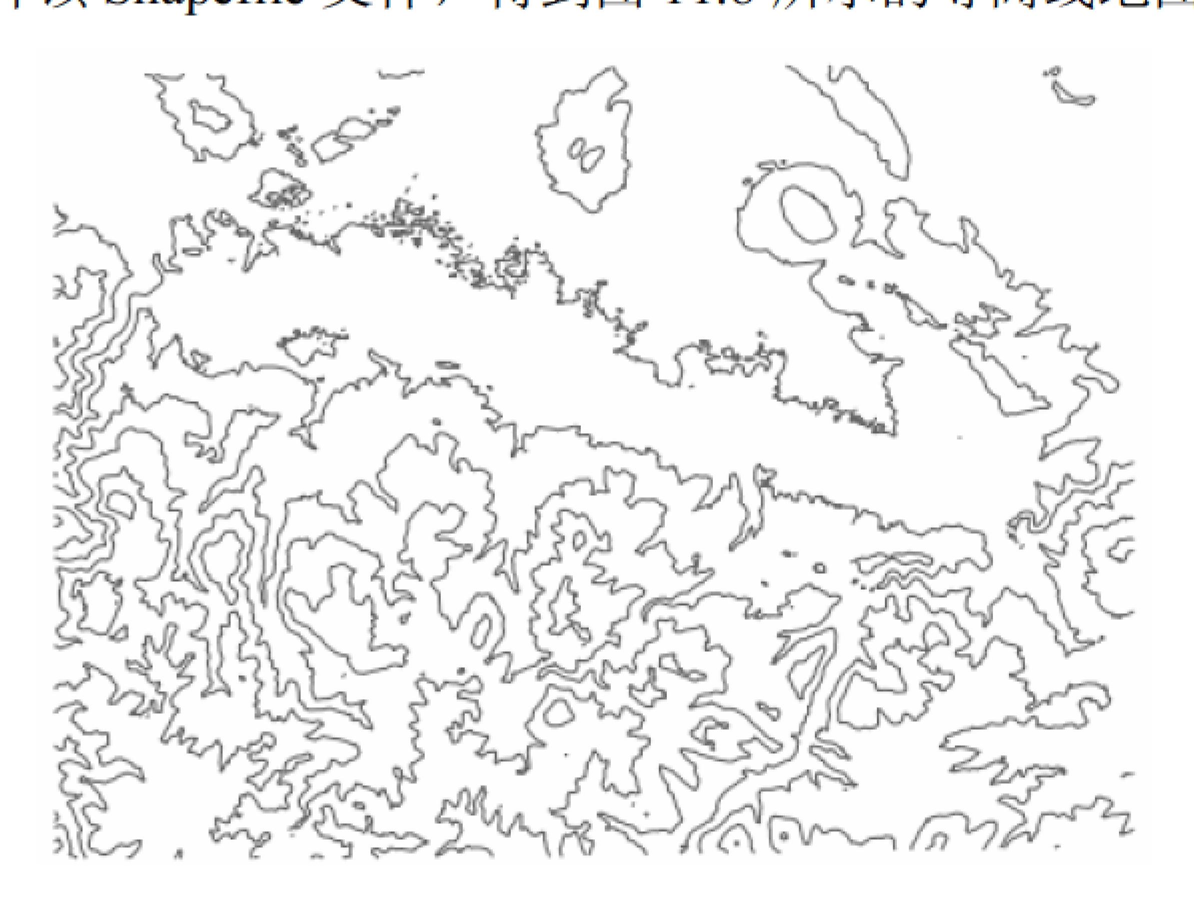


图 11.8　静态生成的等高线矢量数据

11.3.2　动态创建等高线

在前面演示了如何利用 WPS 中的地理处理，从规则格网 DEM 数据获得矢量的等高线。

当前一个发展前沿是直接在样式中应用地理处理，通过着色器转换，动态地将数据显示为地理处理程序处理后的效果。

（1）创建样式文件的基本架构。

新建一个文本文件，将其命名为 contour_dem.sld，在该文件中加入如下样式文件的基本框架内容：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>contour_dem</Name>
    <UserStyle>
      <Title>Contour DEM</Title>
      <Abstract>Extracts contours from DEM</Abstract>
      <FeatureTypeStyle>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

（2）加入着色器转换程序。

着色器转换是通过增加一个 Transformation 元素实现的。在该元素中，指定转换程序的名称以及该转换需要的参数。本实践中使用 gs:Contour 转换程序。

在<FeatureTypeStyle>与</FeatureTypeStyle>之间加入如下内容，实现着色器转换：

```
<Transformation>
  <ogc:Function name="gs:Contour">
    <ogc:Function name="parameter">
      <ogc:Literal>data</ogc:Literal>
    </ogc:Function>
    <ogc:Function name="parameter">
      <ogc:Literal>levels</ogc:Literal>
      <ogc:Literal>1100</ogc:Literal>
      <ogc:Literal>1200</ogc:Literal>
      <ogc:Literal>1300</ogc:Literal>
      <ogc:Literal>1400</ogc:Literal>
      <ogc:Literal>1500</ogc:Literal>
      <ogc:Literal>1600</ogc:Literal>
      <ogc:Literal>1700</ogc:Literal>
      <ogc:Literal>1800</ogc:Literal>
      <ogc:Literal>1900</ogc:Literal>
      <ogc:Literal>2000</ogc:Literal>
    </ogc:Function>
```

```
    </ogc:Function>
  </Transformation>
```

从上面的代码可以看出，Transformation 元素中使用的是 OGC 过滤函数的语法。该元素的内容是一个指定了着色器转换程序名称的<ogc:Function>。转换程序可能需要接受一些参数，这些参数有些是必选的，有些是可选的。参数以“名称/值”对的方式提供。每个参数的名称与值是通过另一个<ogc:Function name="parameter">来提供的，其中第一个<ogc:Literal>包含的是参数的名称，接下来的是参数的值。在上面的代码中，gs:Contour 是转换程序，data 与 levels 分别是两个参数。

（3）加入等高线显示样式。

上面的着色器转换只负责将规则格网的 DEM 转换为矢量等高线，还需要加入等高线显示样式。在上面的代码后面，加入如下内容：

```
<Rule>
  <Name>rule1</Name>
  <Title>Contour Line</Title>
  <LineSymbolizer>
    <Stroke>
      <CssParameter name="stroke">#000000</CssParameter>
      <CssParameter name="stroke-width">0.4</CssParameter>
    </Stroke>
  </LineSymbolizer>
</Rule>
```

（4）加入另一样式区分不同等高线。

上面的代码使得所有的等高线都使用同一较细的黑线来表示。但是当许多线都使用同等宽度同样颜色表示时，很难区分它们。因此接下来需要加入另一个样式规则，将高程值为 300 倍数的等高线用更粗的线表示。而这需要使用 IEEERemainder 过滤函数。

在上面的代码下面，加入如下内容，实现上述描述功能：

```
<Rule>
  <Name>rule 2</Name>
  <Title>Contour Line (300)</Title>
  <ogc:Filter>
    <ogc:PropertyIsEqualTo>
      <ogc:Function name="IEEERemainder">
        <ogc:Function name="int2ddouble">
        <ogc:PropertyName>value</ogc:PropertyName>
        </ogc:Function>
        <ogc:Function name="parseDouble">
          <ogc:Literal>300.0</ogc:Literal>
        </ogc:Function>
```

```
            </ogc:Function>
            <ogc:Literal>0</ogc:Literal>
        </ogc:PropertyIsEqualTo>
    </ogc:Filter>
    <LineSymbolizer>
        <Stroke>
            <CssParameter name="stroke">#662200</CssParameter>
            <CssParameter name="stroke-width">1</CssParameter>
        </Stroke>
    </LineSymbolizer>
</Rule>
```

（5）加入等高线高程值标注。

一个好的地形图应该有高程注记。由于对等高线我们使用了两种样式，因此对于标注最好也使用不同类型的标注。

在上面的代码下面加入如下内容，实现等高线的高程标注：

```
<Rule>
    <Name>rule 3</Name>
    <Title>Label (100)</Title>
    <TextSymbolizer>
        <Label>
            <ogc:Function name="round">
                <ogc:PropertyName>value</ogc:PropertyName>
            </ogc:Function>
        </Label>
        <Font>
            <CssParameter name="font-family">Arial</CssParameter>
            <CssParameter name="font-weight">Normal</CssParameter>
            <CssParameter name="font-size">10</CssParameter>
        </Font>
        <LabelPlacement>
            <LinePlacement/>
        </LabelPlacement>
        <Halo>
            <Radius>
                <ogc:Literal>2</ogc:Literal>
            </Radius>
            <Fill>
                <CssParameter name="fill">#FFFFFF</CssParameter>
                <CssParameter name="fill-opacity">0.6</CssParameter>
            </Fill>
```

```
        </Halo>
        <Fill>
          <CssParameter name="fill">#662200</CssParameter>
        </Fill>
        <Priority>2000</Priority>
        <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="repeat">300</VendorOption>
        <VendorOption name="maxDisplacement">50</VendorOption>
        <VendorOption name="maxAngleDelta">30</VendorOption>
        <VendorOption name="spaceAround">20</VendorOption>
      </TextSymbolizer>
    </Rule>
    <Rule>
      <Name>rule 4</Name>
      <Title>Label (300)</Title>
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:Function name="IEEERemainder">
            <ogc:Function name="int2ddouble">
            <ogc:PropertyName>value</ogc:PropertyName>
            </ogc:Function>
            <ogc:Function name="parseDouble">
              <ogc:Literal>300.0</ogc:Literal>
            </ogc:Function>
          </ogc:Function>
          <ogc:Literal>0</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
      <TextSymbolizer>
        <Label>
          <ogc:Function name="round">
            <ogc:PropertyName>value</ogc:PropertyName>
          </ogc:Function>
        </Label>
        <Font>
          <CssParameter name="font-family">Arial</CssParameter>
          <CssParameter name="font-weight">Bold</CssParameter>
          <CssParameter name="font-size">10</CssParameter>
        </Font>
        <LabelPlacement>
          <LinePlacement/>
        </LabelPlacement>
```

```
            <Halo>
              <Radius>
                <ogc:Literal>2</ogc:Literal>
              </Radius>
              <Fill>
                <CssParameter name="fill">#FFFFFF</CssParameter>
                <CssParameter name="fill-opacity">0.6</CssParameter>
              </Fill>
            </Halo>
            <Fill>
              <CssParameter name="fill">#662200</CssParameter>
            </Fill>
            <Priority>3000</Priority>
            <VendorOption name="followLine">true</VendorOption>
            <VendorOption name="repeat">300</VendorOption>
            <VendorOption name="maxDisplacement">50</VendorOption>
            <VendorOption name="maxAngleDelta">30</VendorOption>
            <VendorOption name="spaceAround">20</VendorOption>
          </TextSymbolizer>
        </Rule>
```

（6）将 contour_dem.sld 上传到 GeoServer 中。

利用 GeoServer 的 Web 管理页面，新建一个名为 contour_dem 的样式，将其样式文件指定为 contour_dem.sld。

（7）设置图层的样式。

按照前面实践介绍的方式，在 GeoServer 的 Web 管理页面中单击“图层”，在图层列表中选择 sfdem，进入编辑图层页面。在该页面的“发布”选项卡中，将 sf 工作区中的 sfdem 图层的默认样式设置为 contour_dem，并将 dem 样式添加到“selected styles”中，如图 11.9 所示。

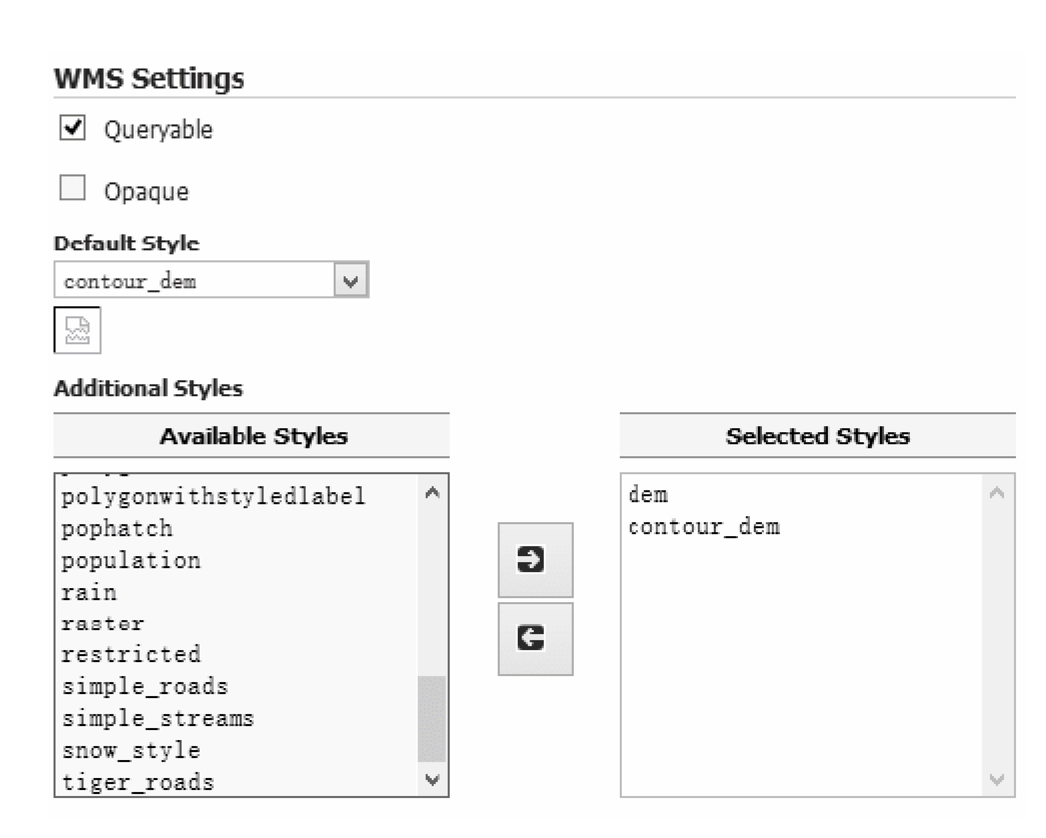


图 11.9 设置图层样式

（8）图层预览。

在 GeoServer 的 Web 管理页面中，使用 OpenLayer 预览 sf:sfdem 图层。不过为了同时显示规则格网的 DEM 以及等高线，需要对 URL 修改两个参数。

- lalyer=sf:sfdem,sfdem
- styles=dem,

修改后的 URL 如下：

```
http://localhost:8080/geoserver/sf/wms?service=WMS&version=1.1.0&request=GetMap&layers=sf:sfdem,sf:sfdem&styles=dem,&bbox=589980.0,4913700.0,609000.0,4928010.0&width=512&height=385&srs=EPSG:26713&format=application/openlayers
```

上述 URL 得到图 11.10 所示的同时以两种样式显示的同一数字高程模型数据。底层是规则格网的 DEM，上层是用地理处理程序动态生成的等高线。

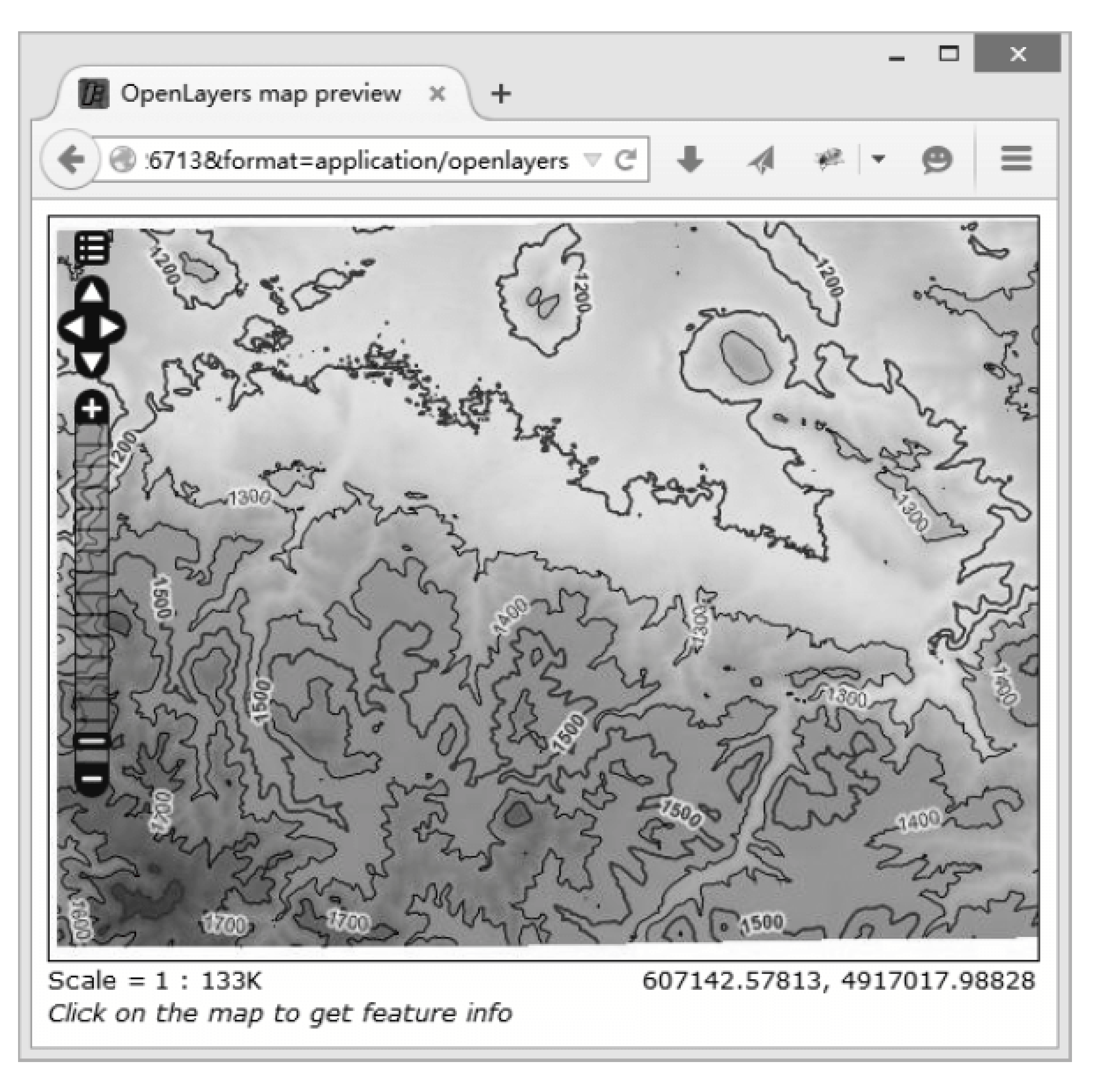


图 11.10　在样式中利用 WPS 动态创建等高线

如果没有得到正确的显示结果，可参考下载文件“Codes\Walkthrough17”文件夹中的 contour_dem.sld 文件。

11.4　实践 18：在 OpenLayers 中使用 WPS

在本实践中，将演示如何使用 OpenLayers 调用 WPS 服务，执行相交与缓冲区计算两个地理处理。

11.4.1 页面设计

新建一个名为 Walkthrough18 的文件夹，在其中加入名为 WpsClient.html 的文件。该文件的内容如下：

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <meta name="apple-mobile-web-app-capable" content="yes">
    <title>OpenLayers WPS 客户端实例</title>
    <style>
        h1 {
            border-bottom: 1px solid #fcb100;
            font-size: 130%;
            margin-bottom: 0.5em;
        }

        .smallmap {
            border: 1px solid #ccc;
            height: 512px;
            width: 512px;
        }
    </style>
    <script src="http://dev.openlayers.org/OpenLayers.js"></script>
    <script src="WpsClient.js"></script>
</head>
<body onload="init()">
    <h1 id="title">WPS 客户端实例</h1>
    <div id="map" class="smallmap"></div>
</body>
</html>
```

11.4.2 代码实现

所有的 JavaScript 代码放在了 WpsClient.js。

（1）新建一个名为 WpsClient.js 的文件。

（2）地图初始化。

在 WpsClient.js 文件中加入如下代码，实现地图初始化工作：

```
var map, client, intersect, buffer;

function init() {
    map = new OpenLayers.Map('map', {
        allOverlays: true,
        center: [114, 16],
        zoom: 4,
        layers: [new OpenLayers.Layer.Vector()]
    });
}
```

（3）加入两个要素。

在 init()继续加入如下代码，实现在地图中加入指定坐标的两个要素：

```
var features = [new OpenLayers.Feature.Vector(OpenLayers.Geometry.fromWKT(
    'LINESTRING(117 22,112 18,118 13, 115 8)'
))];
var geometry = OpenLayers.Geometry.fromWKT(
    'POLYGON((110 20,120 20,120 10,110 10,110 20),(112 17,118 18,118 16,112 15,112 17))'
);

map.baseLayer.addFeatures(features);
map.baseLayer.addFeatures([new OpenLayers.Feature.Vector(geometry)]);
```

（4）创建 OpenLayers.WPSClient 类的实例。

要访问 WPS 服务中的地理处理，首先需要 WPS 服务的 URL 作为参数初始化一 OpenLayers.WPSClient 实例。

在 init()继续加入如下代码，实现实例化一个 WPS 客户端对象：

```
client = new OpenLayers.WPSClient({
    servers: {
        opengeo: 'http://localhost:8080/geoserver/wps'
    }
});
```

（5）创建用于相交计算的地理处理程序。

相交计算的地理处理程序是 JTS:intersection，其输入参数可以是一个要素或一个几何对象，或者是它们的集合。

在 init()继续加入如下代码，实现创建相交计算的地理处理程序，并设置其参数：

```
intersect = client.getProcess('opengeo', 'JTS:intersection');
```

```
intersect.configure({
    // 输入可以是一个要素或一个几何对象，或是它们的集合
    inputs: {
        a: features,
        b: geometry
    }
});
```

（6）创建用于缓冲区计算的地理处理程序。

缓冲区计算的地理处理程序是 JTS:buffer。我们把相交计算得到的结果作为其输入参数。若要执行地理处理，则需要调用 execute 函数。

在 init()继续加入如下代码，执行缓冲区计算。在成功执行返回结果，将结果添加到地图中。

```
buffer = client.getProcess('opengeo', 'JTS:buffer');
buffer.execute({
    inputs: {
        geom: intersect.output(),
        distance: 1
    },
    success: function (outputs) {
        // outputs.result 是一个要素或一个要素数组。
        map.baseLayer.addFeatures(outputs.result);
    }
});
```

（7）运行与调试程序。

用 Firefox 等浏览器直接打开 WpsClient.html 文件，便可得到图 11.11 所示的两个要素，以及两要素相交后得到的两条线段的缓冲区图形。



图 11.11　程序运行结果

程序代码位于下载文件的“Codes\Walkthrough18”文件夹中。

11.5　习　题

（1）在样式中，利用 gs:PolygonExtraction 地理处理程序，动态从 sf:sfdem 图层中提取 1300~1500 米、1500~1700 米以及 1700~1900 米的等高线，并将它们绘制为多边形，结果如图 11.12 所示。参考代码见下载文件的“Codes\Assignment11\ polygons_dem.sld”文件。

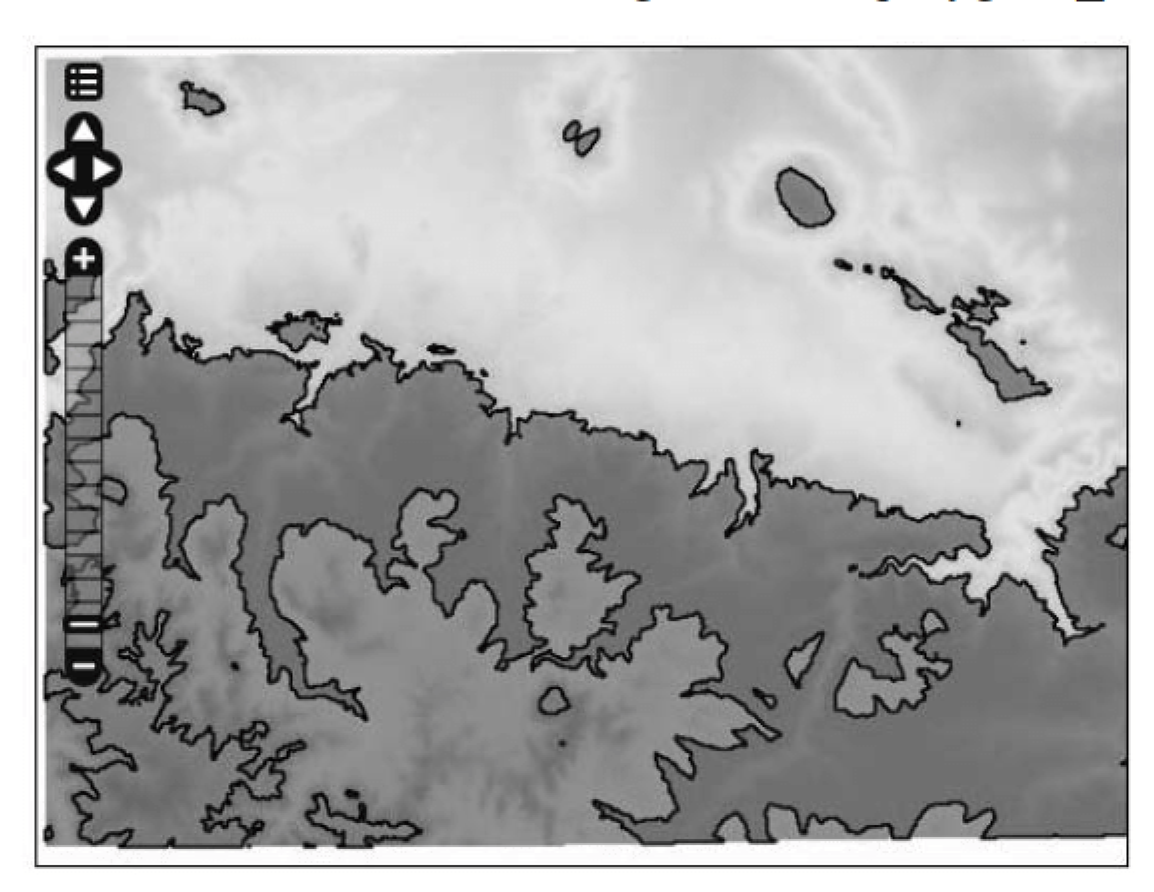


图 11.12　提取等高线并显示为多边形

（2）在样式中，利用 gs:RasterAsPointCollection 地理处理程序，动态从 sf:sfdem 图层中提取每个格网的点，并显示其对应的高程值，结果如图 11.13 所示。参考代码见下载文件的“Codes\Assignment11\ point_dem.sld”文件。

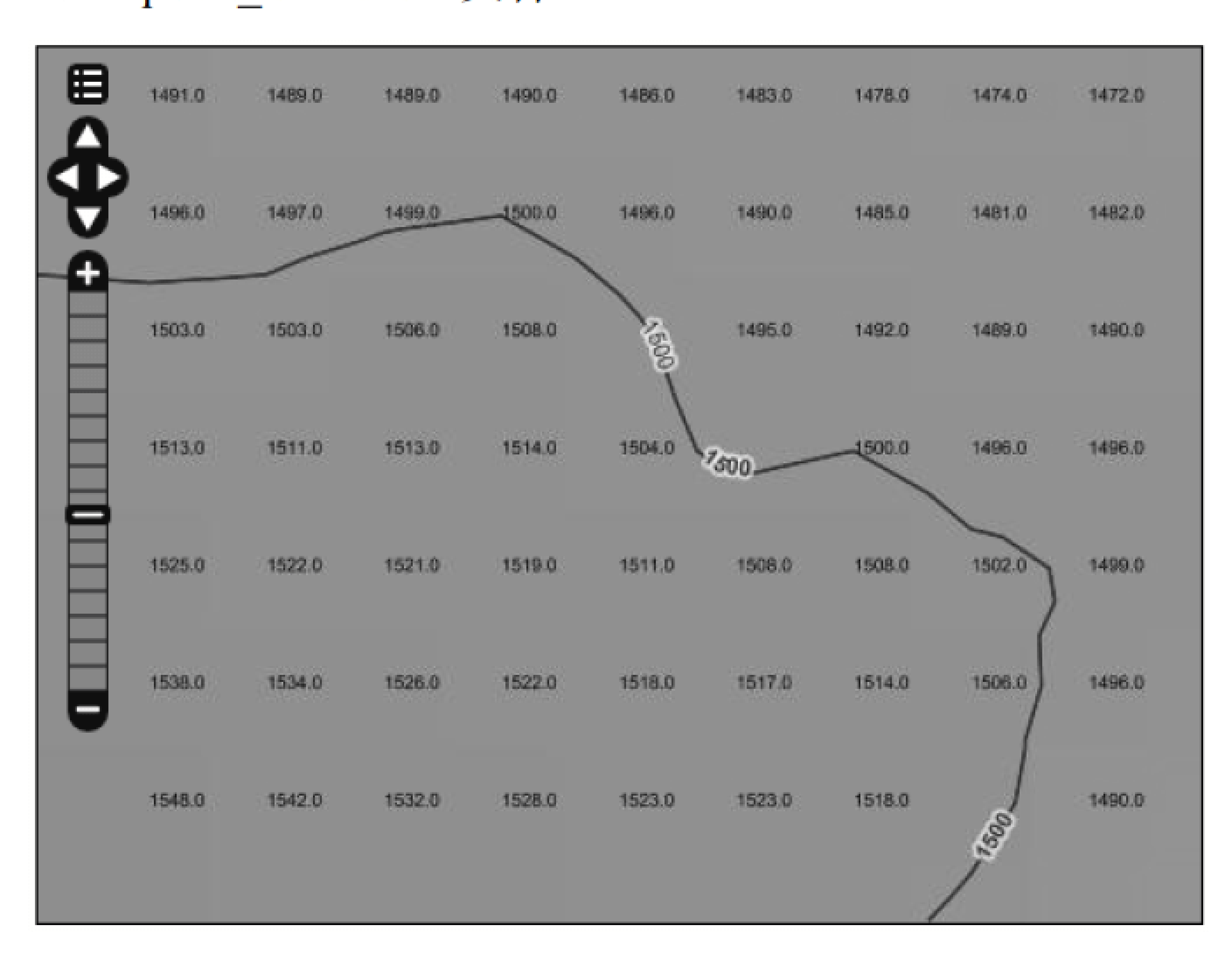


图 11.13　显示规则格网 DEM 中每个格网的高程值

（3）将下载文件“Data\Major_World_Cities”文件夹中的 Major_World_Cities.shp 文件发布为服务。该数据包含了世界主要城市的点数据，其中 POPULATION 字段包含了人口数量字段，通过在样式中利用 gs:Heatmap 地理处理，将该图层动态显示为一栅格形式的热度图。结果如图 11.14 所示。参考代码见下载文件“Codes\Assignment11\heatmap.sld”文件。

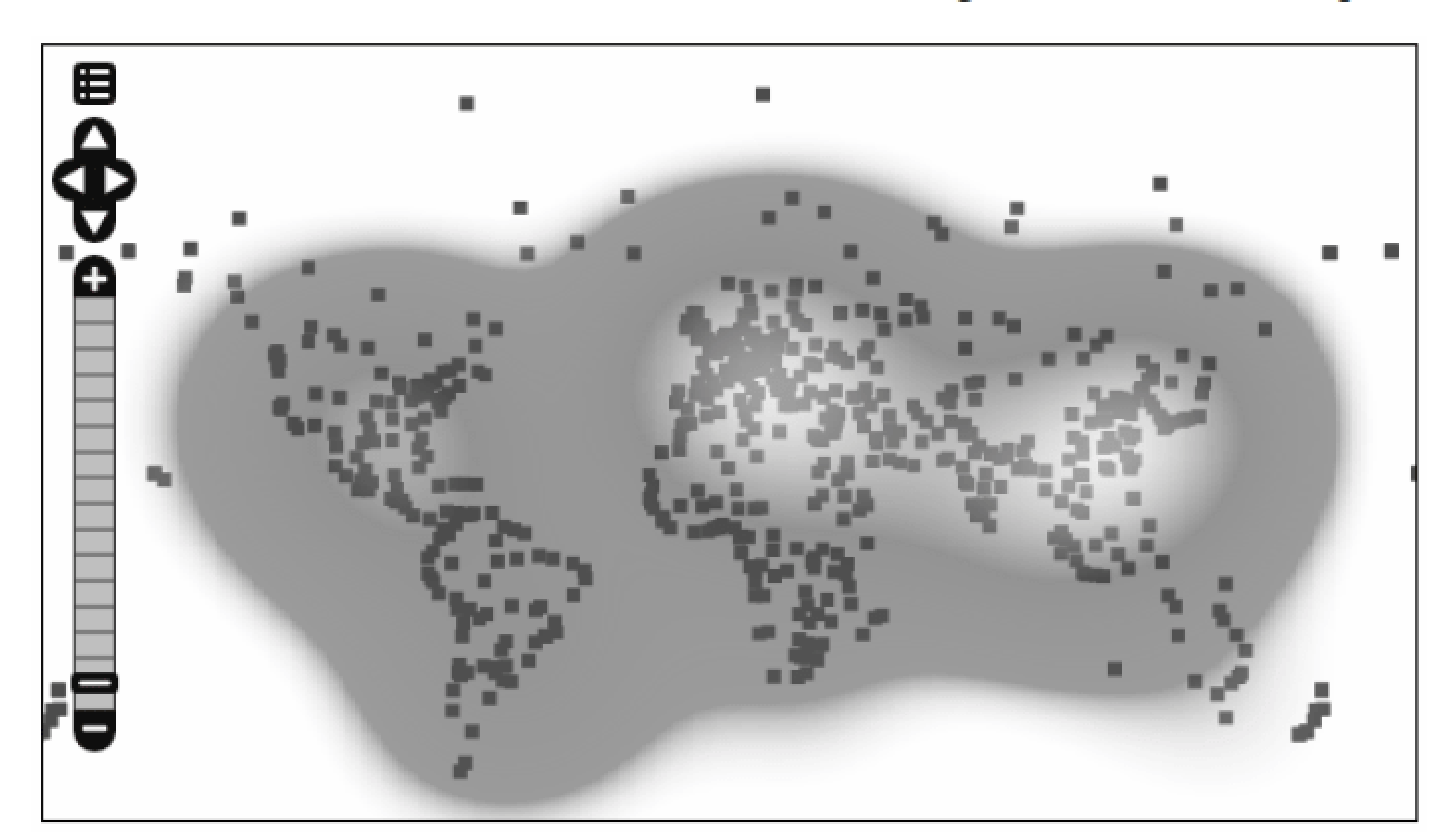


图 11.14　将点图层动态显示为热度图

（4）将下载文件“Data\Certified Vernal Pools”文件夹中的 GISDATA_CVP_PT.shp 文件发布为服务。该数据包含了美国麻省州政府认证的季节性水池点数据。如果使用默认的点符号来样式化，结果如图 11.15 所示。由于点太多，在小比例尺时，许多点聚集在了一起。通过在样式中应用 gs:PointStacker 地理处理，在小比例尺时，将周围点合并显示为一个点，并显示合并的点的数目，如图 11.16 所示。参考代码见下载文件“Codes\Assignment11\Clustered.sld”文件。

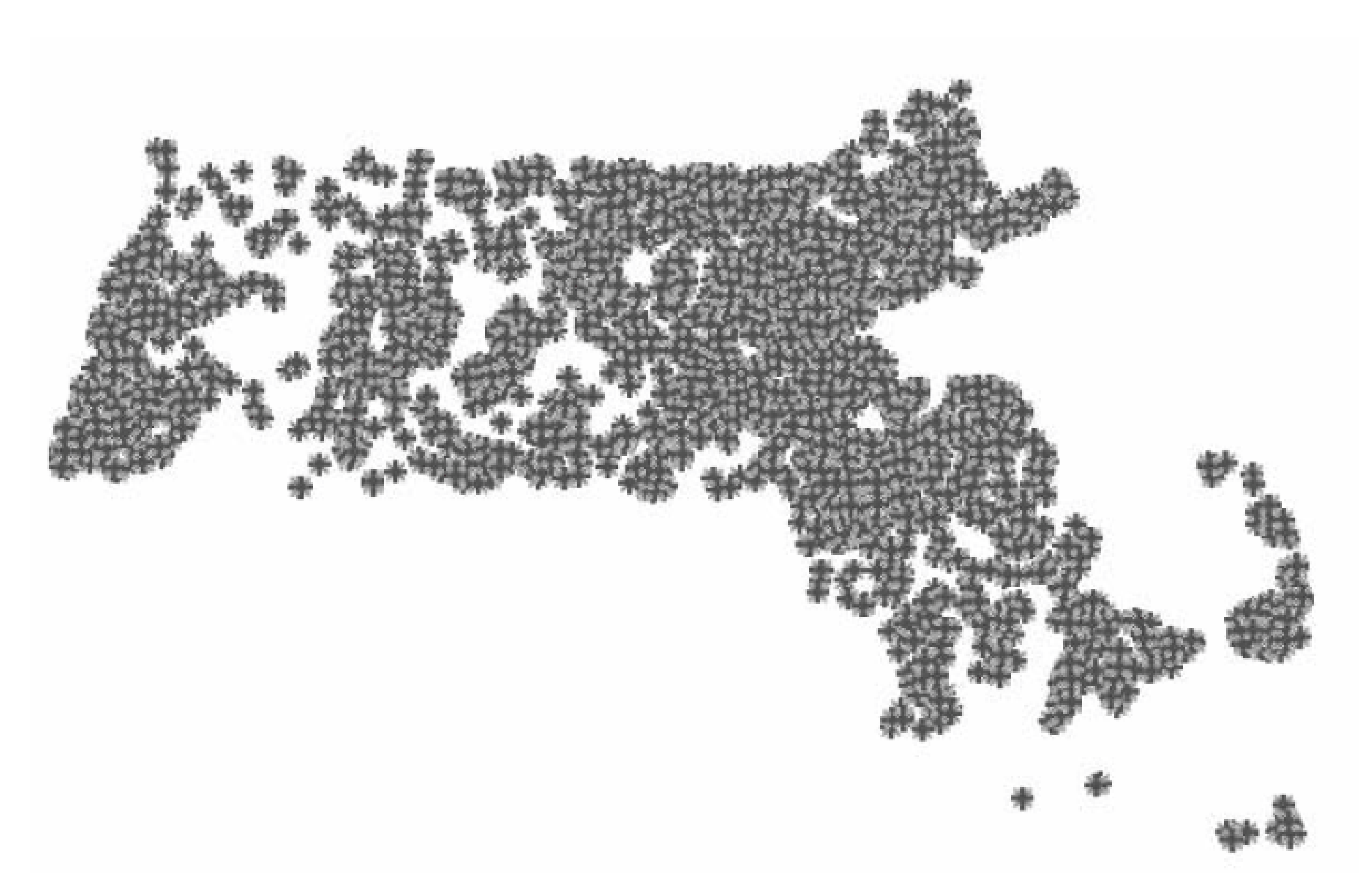


图 11.15　使用默认点符号样式化水池点数据

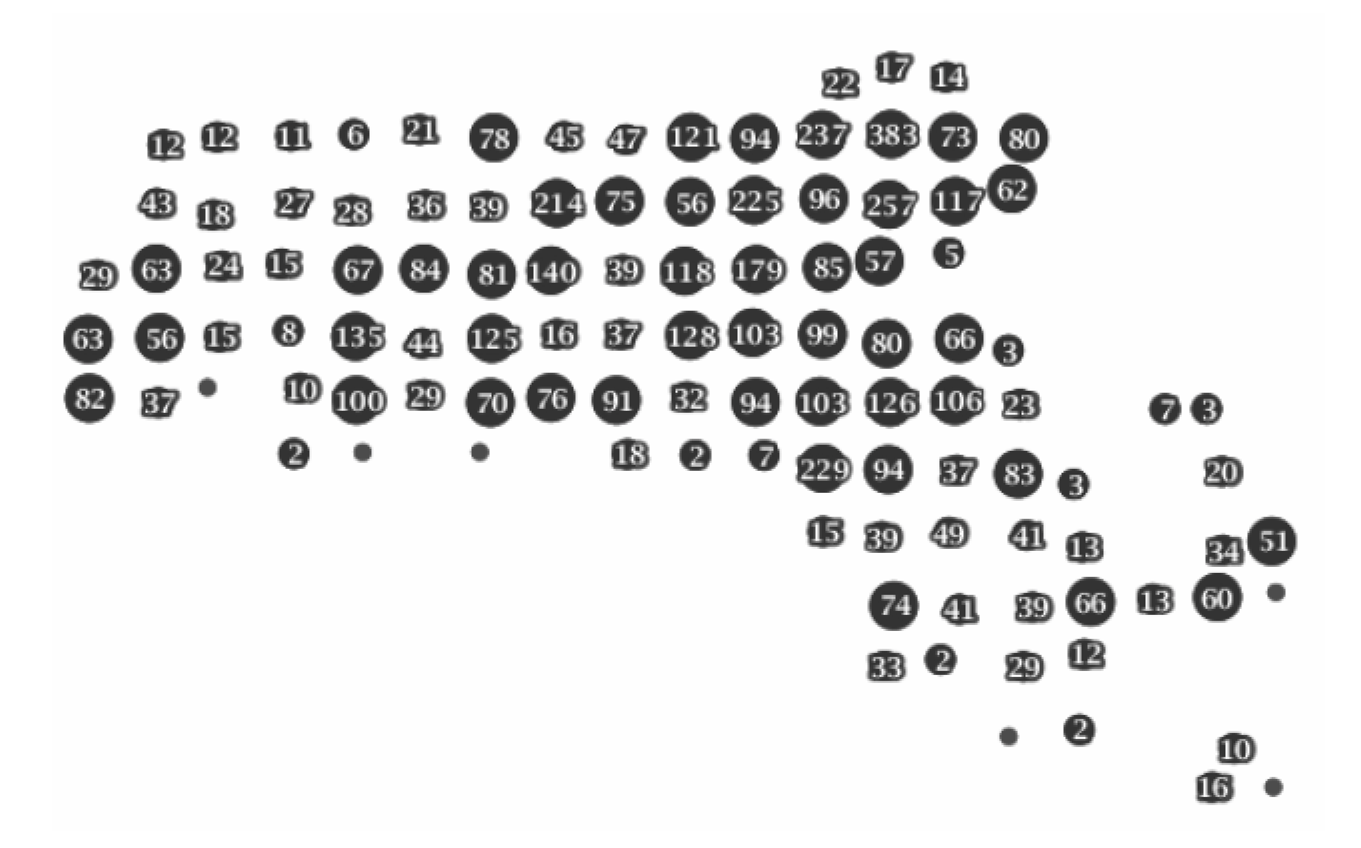


图 11.16　使用 gs:PointStacker 地理处理在小比例尺时显示点的数量

第 12 章

开放数据获取与地图混搭应用

从本章可以学习到：

- 开放数据的方式
- VGI 与众包项目
- OpenStreetMap 及其开放数据的应用
- 地图混搭应用
- 从 OpenStreetMap 获取源数据
- 城市天气预报系统开发

对于一个 GIS 系统或 Web GIS 应用，只有当其中的数据有用时，该系统才有用。正如 GIS 系统中包含了商业软件与开源软件，在空间数据方面，同样存在商业数据与开放数据的区别。软件是由代码和数据共同组成的，“开源”指的只是开放代码，并不包括数据。但当开放代码已经成为共识和现实的时候，新一代的创新者自然又将眼光投向了数据。虽然同为软件的一部分，但开放数据和开放代码却大不相同。开放代码面向的对象仅仅是程序员，也就是说，它停留在技术的层面；但数据的开放，其涉及面却广得多，它不仅和技术人员相关，还与数据的来源、性质以及过去和未来的使用人员都息息相关。

本章将介绍“开放数据”的不同含义，以及创建一个在各种项目中使用最多的开放数据——OpenStreetMap，并将介绍混搭应用及其开发方法。

12.1　开放数据的方式

最近，“开放数据”似乎无处不在。这个词也常常与“政务公开”“众包”“透明政府”和“自由和开放源码软件”等一同出现 。那么，哪些数据算是“开放”数据呢？正如在第 1 章介绍的，不同的组织甚至是商业软件公司，有时会处于某种考虑，也会使用“开源”这个术语。因此，在听到“开放数据”这个术语时，应该考虑到该术语存在许多细微的差别。

根据数据可访问与使用性，可从以下方面来划分不同类型的数据：

（1）公众不能以任何格式查看或下载数据。

（2）只能用静态的方式，例如纸质地图或 PDF 分发数据。

（3）任何人都可以通过一个 Web GIS 系统查看数据，但不能下载数据。

（4）任何人都可以通过 Web 服务方式访问数据，将其集成到自己的 Web GIS 系统中，或用桌面 GIS 系统查看，但不能下载完整的数据集。

（5）数据可免费以商业数据格式下载。

（6）数据可免费以开放格式下载。

当然，组织或个人在宣称自己的数据是开放时，不会像上面那样直白，需要我们仔细甄别。例如：

（1）某个人宣称：“我的数据是开放的，因为我让你看到了数据”，然而由于许可、安全、技术或人力资源的限制，数据本身可能不提供下载。

（2）某一“开放数据门户”可能会提供数据集免费下载，但在门户网站中的一些数据格式可能只能使用商业软件才可读可用。

最开放的数据类型是那些允许完整下载数据、再利用和修改在开放格式。然而，数据开放其他级别毕竟比根本看不见数据要有用。

12.1.1　开放数据许可

即使数据可免费用开放格式下载，该数据仍然可能受许可的限制。针对数据使用的系统

类型（个人、非商业、商业等）以及哪种归属，存在许多类型的开放数据许可。许可也可以规定数据修改的类型，特别是当修改后的数据集再分发时。

这些类型的许可包括知识共享（Creative Commons）、开放式数据库许可证（Open Database License）、开放政府许可（Open Government License）与公众领域（Public Domain）等。《许可开放数据：实用指南》（http://discovery.ac.uk/files/pdf/Licensing_Open_Data_A_ Practical_Guide.pdf）详细介绍了相关概念以及实践指南。

12.1.2 商业软件与开放数据

开源软件在使用开放数据格式方面很突出，但是开源软件并不是创建、共享与使用开放数据的唯一选择。例如 ESRI 也花费了大量的资金与时间来研究如何在 ArcGIS 在线（http://server.arcgisonline.com/ArcGIS/rest/services/）以及 ArcGIS 门户产品中发现并下载开放数据。他们认为，如果商业数据库可以很方便地让公众使用 KML 与 CSV 等流行开放格式下载，那么政府部门用户更愿意使用商业软件来维护他们的数据。

除了 ArcGIS 在线，http://services.arcgis.com 也包含了大量的开放数据。例如在第 11 章习题中利用的世界主要城市数据，就来源于 http://services.arcgis.com/oKgs2tbjK6zwTdvi/ ArcGIS/ rest/ services/ Major_World_Cities。进入该服务的查询页面，将查询条件即“Where”文本框设置为“1=1”，表示获取所有数据，输出字段文本框中设置“NAME,COUNTRY, POPULATION,CAPITAL”，输出格式设置为“GEOJSON”，如图 12.1 所示。

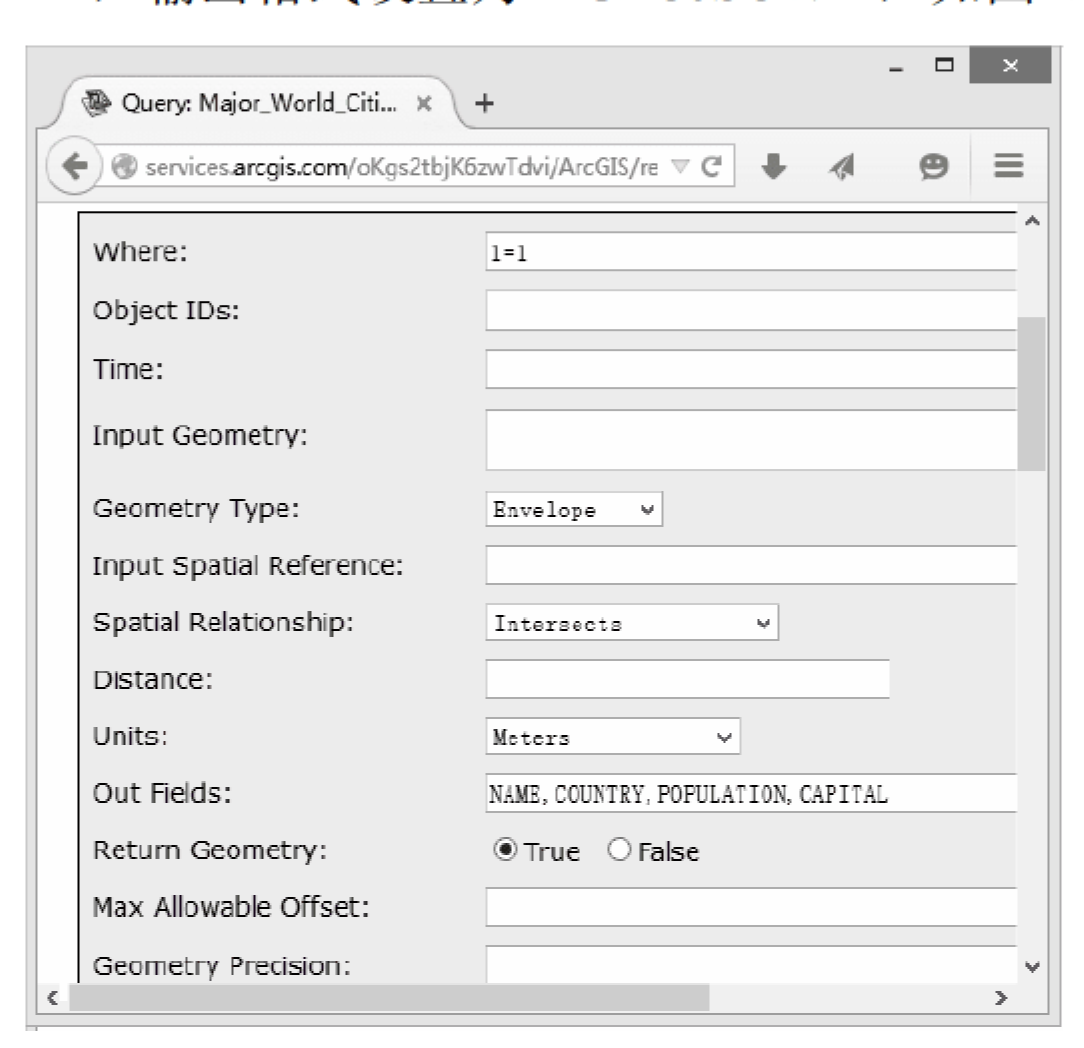


图 12.1　从 ArcGIS 服务中查询开放数据

输入参数之后，在页面底部单击“Query(GET)”按钮，得到图 12.2 所示的以 GeoJSON 格式表示的地理数据。

http://serv...json&token=

n/oKgs2tbjK6zwTdvi/Arc

```
{
  "type" : "FeatureCollection",
  "crs" :
  {
    "type" : "name",
    "properties" :
    {
      "name" : "EPSG:3857"
    }
  },
  "features" : [
    {
      "type" : "Feature",
      "geometry" :
      {
        "type" : "Point",
        "coordinates" : [
          14560306.0338794, 3986163.88626561
        ]
      },
      "properties" : {
        "NAME" : "Kita Kyushu",
        "COUNTRY" : "Japan",
        "POPULATION" : 1525000,
        "CAPITAL" : "N"
      }
```


图 12.2　通过查询得到的 GeoJSON 格式的开放数据

可将该内容保存为一个文件，然后使用 QGIS 等软件打开，便可另存为 Shapefile 等其他格式。

12.2　VGI 与众包项目

如果没有足够的钱或手段来购买所需要的 GIS 数据，或者类似的数据根本就不存在，那么则需要自己收集数据。如果目标是将数据公开，与公众分享所产生的数据，那么则可以考虑在数据收集工作时争取公众参与。 VGI（Volunteered Geographic Information，自发地理信息）和众包是在收集 GIS 数据时争取公众或非领域专家参与的两个概念。

12.2.1　VGI

VGI 在地理信息科学领域已经成为一个炙手可热的名词。该概念由 M.F. Goodchild 于 2007 年提出，是指由大众自愿创建、编辑、管理、维护的地理信息。这种数据是由市民作为传感器，来收集有关周围世界的信息，然后常通过一个已简化的不需要专门特殊培训的用户界面，将数据录入到一个集中的 GIS 数据库。此概念是用户创建内容（User-generated content，简写为 UGC）、Web 2.0 等思想与地理信息系统相结合的产物，反映了互联网时代地理信息新的获取与应用方式。

Openstreetmap、Google Map Maker 都是以采集与管理自发地理信息为基本模式的网络协

作项目，它们提供基础地图，并允许用户创建新的地理数据或更新已有数据。此外，各国政府也都在评估建立“市民举报”的应用程序的可能性，该应该程序允许任何人上传有关城市中的违法违规信息，以引起地方当局的注意。

VGI 被认为是公众参与的地理信息系统在互联网下的发展产物，它提供了普通人操作和使用地理数据的权利，同时为维护动态更新的开放地理数据库提供了可行的方案。

12.2.2　众包

众包是使用公众的力量来收集数据量巨大、一致的数据，或者使用其他方式采集则要花费巨大的数据。例如编写一部百科全书，包含有使用 250 种语言的 3000 万篇文章，如果不采用众包的方式，试想需要雇佣多少人？而维基百科 Wikipedia 网站则仅仅通过众包的方式，实现了该伟大创举。其他众包应用包括寻找丢失的人或车辆的遥感图像应用（edition.cnn.com/2014/03/11/us/malaysia-airlines-plane-crowdsourcing-search/）、根据船舶日志记录旧天气以便创造气候数据库项目（www.oldweather.org/），以及根据人口普查创建可搜索的家谱索引项目（https://familysearch.org/indexing/）等。中国众包案例包括人人猎头、微差事、易到用车等。

众包最适用的是那些机器不能实现而需要人类认知元素的任务。亚马逊甚至还通过它的 Mechanical Turk（aws.amazon.com/mturk/）的服务提供众包服务。通过该网站，通常可以用很少的费用来雇佣一大批不知名的个人来完成某一任务。

众包的概念很符合 VGI，特别是对于要求在很短时间内收集大量数据。但是，并不是所有 VGI 项目都使用众包。一些 VGI 项目侧重于从小样本人群或领域专家收集信息。

12.3　OpenStreetMap 及其开放数据的应用

OpenStreetMap（OSM）是通过 VGI 与众包方式创建的全球电子地图数据库。OSM 由非营利的 OpenStreetMap 基金会支持。OSM 中的数据在开发许可（www.openstreetmap.org/copyright）下，可以免费查询、下载与修改。

OSM 的运行模式与维基百科类似，几乎所有的功能都可以由用户社区的任何成员编辑。OSM 在 2004 年 7 月由 Steve Coast 创立，在 2013 年年初注册用户就已经超过 100 万。虽然只有很少一部分人频繁地编辑地图，不过在某地地点，其数据的详细程度与准确程度，完全不逊于政府部门与商业公司的“权威”数据。在西欧和美国的一些地区尤其如此。图 12.3 展示的是美国宾夕法尼亚州州立大学校园，从中可看出 OSM 包含许多复杂的要素。

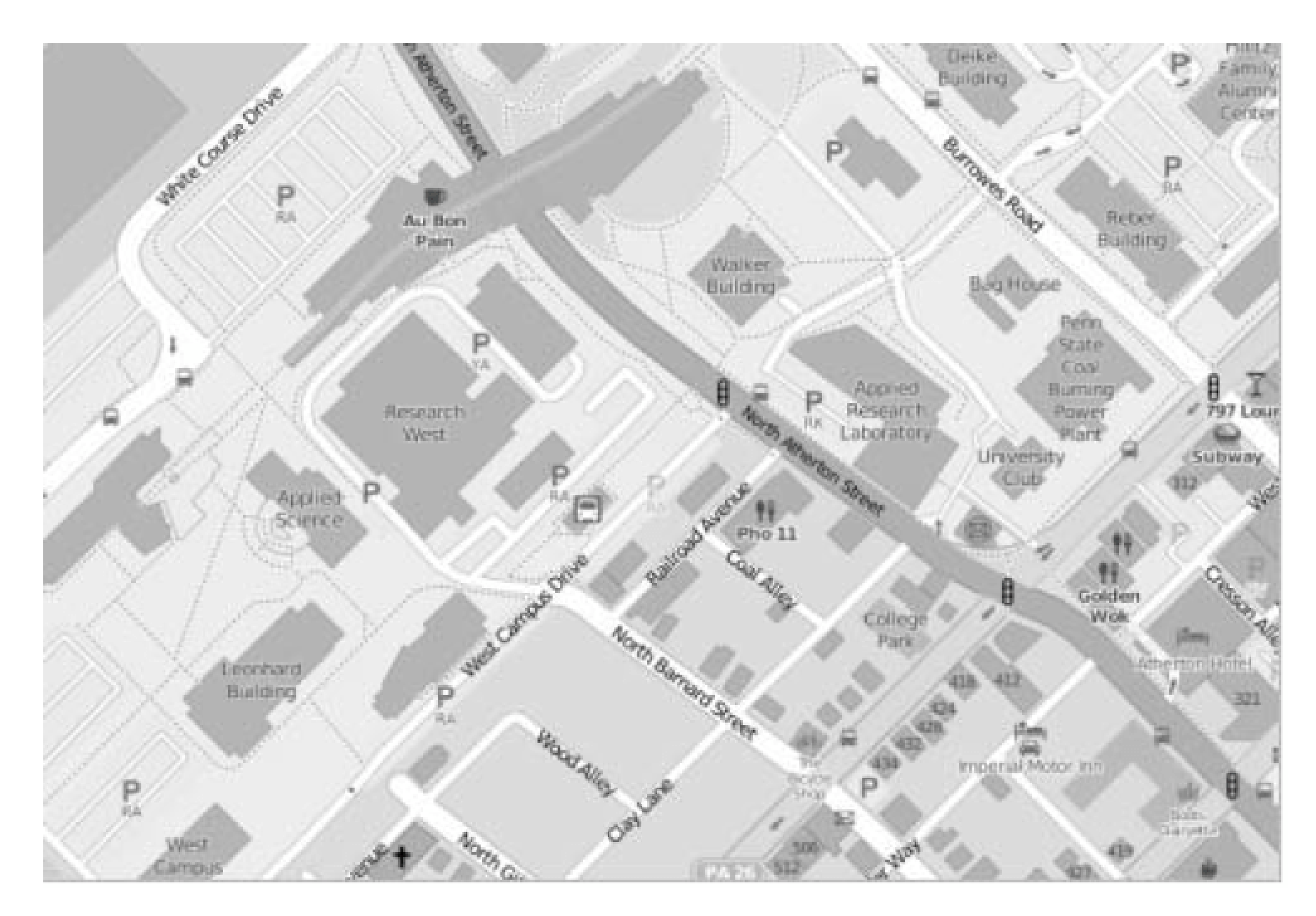


图 12.3　OSM 地图样例

OSM 最初流行于那些数据不能从地方政府免费获取但 GIS 社区却很繁荣的地方。例如在 21 世纪初，英国陆军测量局的数据需要付费购买，那么 OSM 则迅速成长为一个有吸引力的免费替代品。对于那些政府愿意免费共享他们数据的地方，OSM 通常直接使用这些数据。例如，由于将美国人口普查 TIGER 街道数据批量上传到 OSM 中，因此在美国有相当深入的道路覆盖数据。

12.3.1　OpenStreetMap 数据模式

在 OSM 中贡献要素时，虽然仍然使用的是点、线、面几何图形，但是 OSM 却使用了与普通 GIS 很不一样的数据模型。

OSM 采用的是准拓扑的数据模型，其中最基本的是 Node、Way 与 Relation 这 3 个对象，三者关系如图 12.4 所示。

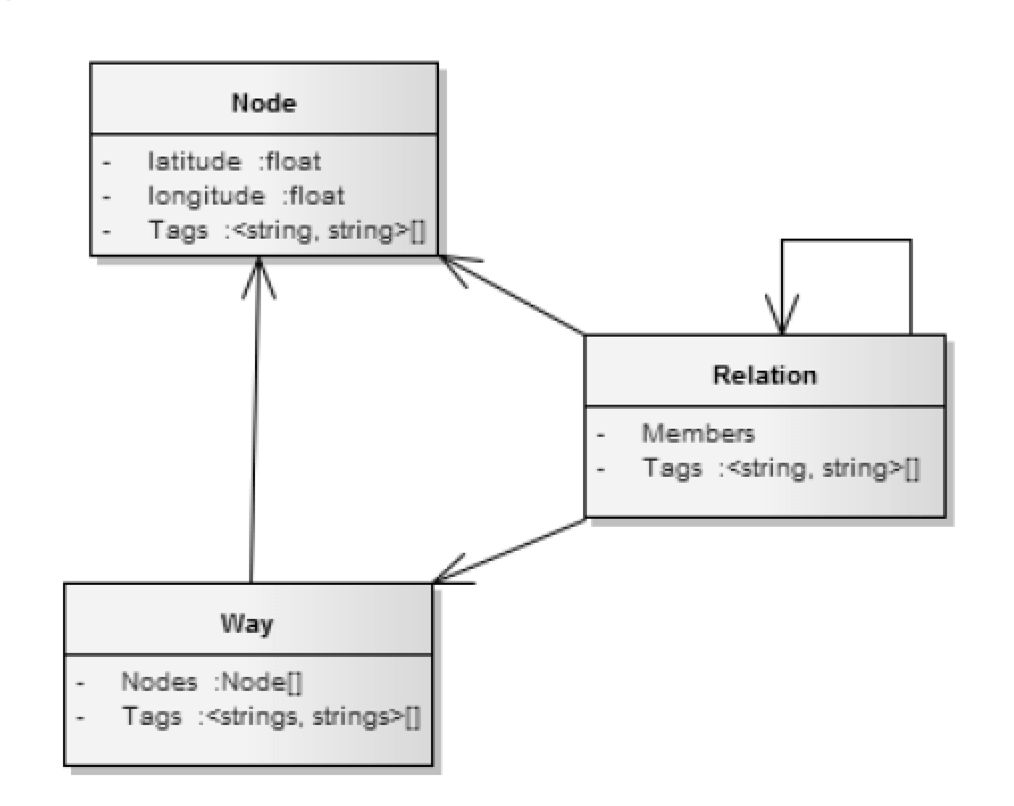


图 12.4　OSM 数据模型中最基本的 3 个对象及其关系

Node 表示一个节点，也就是空间中的点，比如说一个景点或者山峰，对应 GIS 中的点要素。它是 OSM 数据类型中唯一用来标识位置信息的类型，Way 与 Relation 都依赖于 Node。下面是一个 Node 的例子：

```
<osm version="0.6" generator="OpenStreetMap server">
    <node id="483034256" lat="55.9458449" lon="-3.2035477" version="1"
        changeset="2369219" user="spytfyre" uid="166957" visible="true"
        timestamp="2009-09-04T13:35:42Z">
        <tag k="name" v="The Blue Blazer" />
        <tag k="amenity" v="pub" />
    </node>
</osm>
```

每个节点除了表示唯一标识的 id 之外，至少还需要包含经度和纬度两个属性。其他主要可选属性包括：

（1）version：表示该对象的版本；

（2）changeset：标识变化（即由同一个用户执行的批量插入/更新）的版本号；

（3）user：提交更改用户的昵称；

（4）uid：提交更改用户的唯一标识 id；

（5）timestemp：更改时间戳；

（6）其他任意的属性信息：使用标签（tag）元素来增加，该元素中使用“名称/值”对的方式，k 表示属性的名称，v 表示属性的值。例如上面的例子中，该要素包括 name 与 amenity 两属性，其值分别为“The Blue Blazer”与“pub”。

标签元素中的属性名称可以是任意的，但是有一个标签规范（wiki.openstreetmap.org/wiki/Map_Features）规定了最常用的要素属性名称。

Way 表示有序排列的节点，以折线的形式呈现，如果是封闭圈的话，那么可能是以多边形的方式呈现。这类原始数据大多用来呈现为街道、河流或一个区块，比如说森林、公园、停车场或者是一片湖泊。下面是一个 Way 的例子：

```
<osm version="0.6" generator="OpenStreetMap server">
    <way id="43157302" visible="true" timestamp="2009-10-26T10:45:09Z"
        version="1" changeset="2954960" user="Ed Avis" uid="31257">
        <nd ref="540653724" />
        <nd ref="25507043" />
        <nd ref="107762" />
        <nd ref="25507038" />
        <nd ref="107759" />
        <tag k="highway" v="primary" />
        <tag k="lcn  ref" v="6a" />
        <tag k="name" v="Parliament Street" />
    </way>
```

```
</osm>
```

其中 nd 表示一个节点的引用，其 ref 指向的是节点的 id。

Relation 表示关系，是用来表示各个原始数据（节点与节点、节点与道路、道路与道路）的关系，比如说道路与道路的拐弯限制，一条道路分叉出来的各种小道路，或者一个区域中的一个洞等。下面是一个 Relation 的例子：

```
<osm version="0.6" generator="OpenStreetMap server">
    <relation id="113421" visible="true" timestamp="2009-11-03T10:08:27Z"
        version="2" changeset="3023369" user="Jonathan Bennett" uid="5352">
        <member type="node" ref="270186" role="via" />
        <member type="way" ref="4418767" role="from" />
        <member type="way" ref="4641665" role="to" />
        <tag k="restriction" v="no_right_turn" />
        <tag k="type" v="restriction" />
    </relation>
</osm>
```

上述实例表示从哪条道路经过某个节点到哪条道路去，并指明在该节点处禁止右拐。

12.3.2　OpenStreetMap 的使用

OSM 最基本或最常用的使用方式是获取其地图切片，将其作为其他专题图层的基础底图。以这种方式使用 OSM 的高知名度的网站包括 Foursquare、Craigslist 与 Wikipedia。在 Google 地图 API 引入潜在费用以后，很多网络开发人员转而使用 OSM 作为基础底图。

从技术角度来说，任何人都可以使用类似 Mapnik 的渲染引擎，将 OSM 数据绘制成地图切片。事实上，本书的第 5 章的实践部分演示了如何实现该任务。图 12.5 显示的是直接使用 OpenStreetMap.org 中的地图时，可选择的基础底图类型。其他像 MapQuest（developer.mapquest.com/web/products/open/map）与 Mapbox（www.mapbox.com/ data-platform）等公司通过 Web 服务的方式也提供了 OSM 地图。



图 12.5　OpenStreetMap.org 提供的不同类型的基础底图

在 2010 年海地大地震后，OSM 树立了一个让非政府组织和国际组织合作的榜样。在短短两天内，OpenStreetMap 和 Crisis Commons 的志愿者使用了卫星映像在 OSM 完成了标记海地 Port-au Price 区域的道路、建筑物和避难营，后来使 OSM 成为“最齐全的海地数字地图”。在海地灾情过后，OSM 也在后来的发生灾难的地区产生了重大的作用。马利（2013 年 1 月）、菲律宾的海燕（2013 年 11 月），还有西非的埃博拉病毒（2014 年 3 月），再次显现了各个不同的国际组织即使通过互联网，也能从 OSM 地图中来帮助人道主义组织进行援助。

此外，OSM 有一个最大的优势是其灵活性，通过已经存在的标签以及基于社区的标签提议机制，可存储任何类型的要素。有时还可以根据要素的子类型来创建专门的专题地图，例如：

（1）OpenCycleMap（www.opencyclemap.org），专门用于绘制自行车道；

（2）OpenSkiMap（openskimap.org），展示滑雪缆车和滑道；

（3）Wheelmap（www.wheelmap.org），用以浏览和标记轮椅无障碍地点；

（4）Philly Fresh Food Map（www.geovista.psu.edu/phillyfood），显示在美国宾夕法尼亚州费城城市农业资源和新鲜食品经销网点，如图 12.6 所示。



图 12.6　费城农业资源和新鲜食品经销网点分布专题图

在这些地图中，OSM 充当了了解当地有用知识的可自由访问的数据库。其中一些地图要素为社区提供了很大的价值，但是这些地区往往无利可图，因此商业地图只提供了很少的信息或根本就没有信息。为了避免地图图面过于杂乱，在默认的 OSM 切片地图中并没有完全显示其数据库中的所有类型的要素，但是这些要素对于那些开发人员往往很珍贵，他们可从中提取自定义的子集并形成专题地图。

要记住的是，正是因为 OSM 允许免费下载并重新利用这些数据，才能有上述这些专题地图。从 OSM 中提取专题地图的网站，通常基于 OSM 的查询 API，称为 XAPI。该 API 通过 Web 服务提交自定义的标签查询。相比下载某地区整个 OSM 的数据集，通过 Web 服务获取匹配标签的要素要实用得多。在本章的实践中将介绍 XAPI 的简单使用。

12.4　地图混搭应用

在互联网进入 Web 2.0 时代后，众多地图站点例如谷歌地图、必应地图和我国的天地图，都为使用者提供了可供调用的 API 接口，我们可以利用这些 API 接口将所需的各类服务整合起来，实现内容的集成。这就是地图混搭应用。

12.4.1　混搭应用的概念

对于混搭技术，目前有个比较统一的定义：混搭是通过其他站点对外提供 API 接口调用，来为特定站点带来新的功能或增加新的内容，即从多个分散的站点获取信息源，组合成新的网络应用的一种站点模式。混搭的产品形式有很多种，既可以是一家服务商把自己的多个产品或多个功能模块通过各自的 API 接口在自己的平台实现统一的服务整合；也可以是服务商搭建一个通用的平台，将其他服务商的服务转化成统一的服务接口，供用户在平台上自由组合调用。

在地理信息混搭方面有着许多应用。特别是因为 Google 等公司推出属于自己的 API，降低以往开发电子地图的门槛，让许多以 Google 地图等电子地图为显示底图的应用网站如雨后春笋般诞生。Programmableweb 网站上列出了超过 1400 个地理信息融入式应用（www.programmableweb.com/tag/mapping）。最为成功的是 24 岁的 Adrian Holovaty，他把美国芝加哥警察局的犯罪统计信息覆盖在 Google 地图上（www.chicagocrime.org）。这样，人们在地图上就可以精确查明 30 天的时段内发生性侵犯犯罪的地点。在地图上，每一个犯罪地点都用一个图钉符号标出，芝加哥人能迅速获知应该避开哪些危险的火车站和街区。社区活动家 JamesCappleman 对 Holovaty 的芝加哥犯罪网印象深刻，因为这样居民们就不会再轻信那些街区安全的说法了。而包括旧金山在内的其他一些城市希望 Holovaty 也能为他们开发犯罪定位网站。同样，佛罗里达性犯罪网（MapSexOffender.com）把 Google 地图和被宣判的性犯罪者的资料结合起来。访问者可以调阅所在社区地图，单击图标查看每一个犯罪者的姓名、最新地址和照片。而美国的驾车者如果要找最便宜的加油站，只需要单击结合了 Google 地图和汽油伙伴网站（Gas-buddy.com）加油站价格的数据库的链接就可以了。同样的，购房者可以利用 Google 的地图，精确查明适合的房源地点。以搜索房源的 Housingmaps.com 网站为例，当 Google 地图刚发布，电脑动画工程师 Paul Rademacher 随即开发了 Housingmaps.com，他将 Google 地图和全美所有在 Craigslist 上公布的公寓名单对接。此外还有提供飞机航班即时信息的 fboweb.com，结合天气信息的 Weather Underground 等。

12.4.2　网络资源

构建混搭应用最重要的是有好的创意，此外找到相应的网络资源也是必不可少的条件。可以从 www.programmableweb.com/tag/mapping 寻找相应的 API 与数据。这里简单介绍几个

重要的常用网络资源。

12.4.2.1 GeoNames

GeoNames 是一个免费的全球地理数据库。GeoNames 的目标是把各种来源的免费数据进行集成并制作成一个数据库或一系列的 Web 服务。GeoNames 的用户包括 Microsoft Popfly、Slide.com、LinedIn 和 Tagzania。

GeoNames 地名辞典包含了 650 万个地点、将近 200 种语言的 850 万个地名和 200 万种别名。所有特征均分门别类地放入 9 个特征类，这些特征类再细分为 645 个特性代码。地理信息还详细到坐标、行政区划、邮政编码、人口、海拔和时区。GeoNames 的数据收集自（美国）国家测绘机构、国家统计署、国家邮政局，还有美国陆军。数据可通过一系列 Web 服务和数据库导出免费使用。并且 GeoNames 采用了维基百科的风格允许用户纠正数据和添加新的地名。2007 年 4 月，GeoNames 报告宣称已有 15000 次数据库下载和 3000 万次 Web 服务查询。

GeoNames 回答了诸如此类的问题：这个地方在哪儿？它的坐标是多少？它属于哪个地区或哪个省？有哪些城市或地址靠近这个给定的经纬度？GeoNames 的用户一般利用这些数据或服务计划旅游、房地产、在线社区、商店定位、交通工具追踪、地址确认或关于地理地图方面的应用。

也可以通过其丰富的 Web 服务来使用 GeoNames。它是基于 REST 风格设计的，所以可以使用任何用来从 URL 请求数据的代码。在 GeoNames 上所能执行的查询的种类很多。如下所示的是其中一些例子：

（1）找到某个邮编所代表位置的周边地区，按国家返回（返回 XML 文件或 JSON 提要）。

（2）找到接近给定纬度/经度的邮编（返回 XML 文件）。

（3）找到具有给定地理特征的子集（例如，某个国家的省、该省管辖的各地区，返回 XML 文件或 JSON 提要）。

（4）找到接近给定纬度/经度、邮编或地名的相关维基百科（Wikipedia）文章（返回 XML 文件或 JSON 提要）。

（5）找到某个国家的所有邻国（返回 XML 文件或 JSON 提要）。

（6）找到由 4 个纬度/经度对所确定的地理范围内的全部气象台及其最新的气象观测（返回 XML 文件）。

（7）获得给定纬度/经度所在的时区（返回 XML 文件或 JSON 提要）。

（8）获得代表陆地地区的纬度/经度所对应的以米为单位的海拔（返回 XML 文件或 JSON 提要）。

12.4.2.2 Flickr 的相册服务

Flickr 相册服务是 Web 2.0 应用方式的绝佳例子，它提供了具备社会性网络的相片共享服务，让用户图片的上载、组织和查找都变得异常简单。对于开发者，Flickr 还提供了读写照片数据的公共 API，可以通过 HTTP 的 API 请求发送来调用 Flickr 提供的服务。

Flickr 集合了借由用户间的关系彼此相互连接的数字图像，图像可依其内容彼此产生关系。图片上传者可自己定义该相片的关键字，也就是“标签（Tags）”，如此一来搜索者可很快地找到想要的相片，例如指定拍摄地点或照片的主题，而创作者也能很快了解相同标签下有哪些由其他人所分享的照片。Flickr 也会挑选出最受欢迎的标签名单，缩短搜索相片的时间。Flickr 被普遍认为是有效使用分众分类法（Floksonomy）的范型。此外，Flickr 也是第一个使用标签云（Tag Cloud）的网站。Flickr 也让用户能将照片编入“照片集（Sets）”，或是将有相同标题开头的照片结成组群。然而照片集比传统的文件夹分类模式更有弹性，因为一张照片可被归类到多个照片集中，或是仅分至一个照片集中，或是完全不属于任何的照片集。

可使用如下格式的网址，从 Fickr 中得到带有地理标签的相片（即 GeoRSS 提要）：

http://www.flickr.com/services/feeds/geo/两个字母的国家代码/城市名称

例如，要得到来自中国的 GeoRSS 提要，只需要在网址栏中输入如下地址：

http://www.flickr.com/services/feeds/geo/cn

而要得到北京的相片的 GeoRSS 提要，可使用如下网址：

http://www.flickr.com/services/feeds/geo/cn/Beijing

默认返回的是 GeoRSS 格式的提要。GeoRSS 提供了一种在 RSS (或者 Atom) 种子里通过特定的编码来包含地理参考信息的方法。嵌入 GeoRSS 非常简单，仅仅在每个条目中增加一个类似 <georss:point>45.256 -71.92</georss:point> 这样的元素就行了，这里使用的是简易 GeoRSS 格式，如果需要复杂完整的编码格式，可以选择支持更多特性的 GeoRSS-GML 格式。这两种 GeoRSS 格式都支持基本的地理特征（点、线、边框和多边形）。和 KML 一样，商业化的地图 API 和开源地图 API 都支持 GeoRSS，并且主要作为导入数据的格式使用。GeoRSS 许诺对整合内容的会有更好的支持。

当然，对于使用 JavaScript 来访问，最简单的方式是使用 JSON 格式。要得到 JSON 格式的提要，只需要在参数中加入格式定义即可，例如下面的地址可得到全球范围内以 JSON 格式返回的相片提要：

http://www.flickr.com/services/feeds/geo?format=json

可在地址中加入标签定义来缩小搜索范围，例如：

http://www.flickr.com/services/feeds/geo?format=json&tags=technology

12.4.2.3　Yahoo!天气

Yahoo!天气提供了未来 5 天的天气预报，以及每天的详细天气。在 JavaScript 中使用 Yahoo!天气服务一种最简单的方式是使用 YQL 查询其 weather.forecast 表。

YQL（Yahoo! Query Language）是 Yahoo!发布的一种支持对互联网上的数据进行查询、过滤、连接，且类似 SQL 语法的简单语言。用 YQL 官方的话来说，有了 YQL，开发人员只需要使用一种简单的查询语言即可访问和操控互联网上丰富的数据，而不再需要反复学习使用各种各样的 API。YQL 为我们提供丰富、实时的方法来操控互联网上任意可访问的 API。

YQL 就像一个超大的数据库。从理论上讲，这个数据库可以包含整个互联网上的信息。无论是要基于各种 API 操作数据，还是从 feed 源（如 RSS、XML、ATOM）获取数据，甚至

是从指定的 HTML 页面上抓取结果,所需要的就是使用 YQL 这种类似 SQL 的简单查询语言:

SELECT something FROM table_name WHERE some_field=some_value

YQL 能够以规范的格式(XML/JSON)将结果返回。

Yahoo! 同时向开发者提供了一个可视化的 Web 控制台，网址为 https://developer.yahoo.com/yql/console/，让我们在使用和调试 YQL 的过程中更加高效便捷。例如要通过名称查询其对应的地理坐标，类似于前面介绍的 GeoNames，便可以在 Web 控制台的“YOUR YQL STATEMENT”中输入类似如下的语句:

select * from geo.places where text='Beijing China'

然后单击“Test”按钮，便可在控制台的“Formatted View”中显示查询结果，如图 12.7 所示。

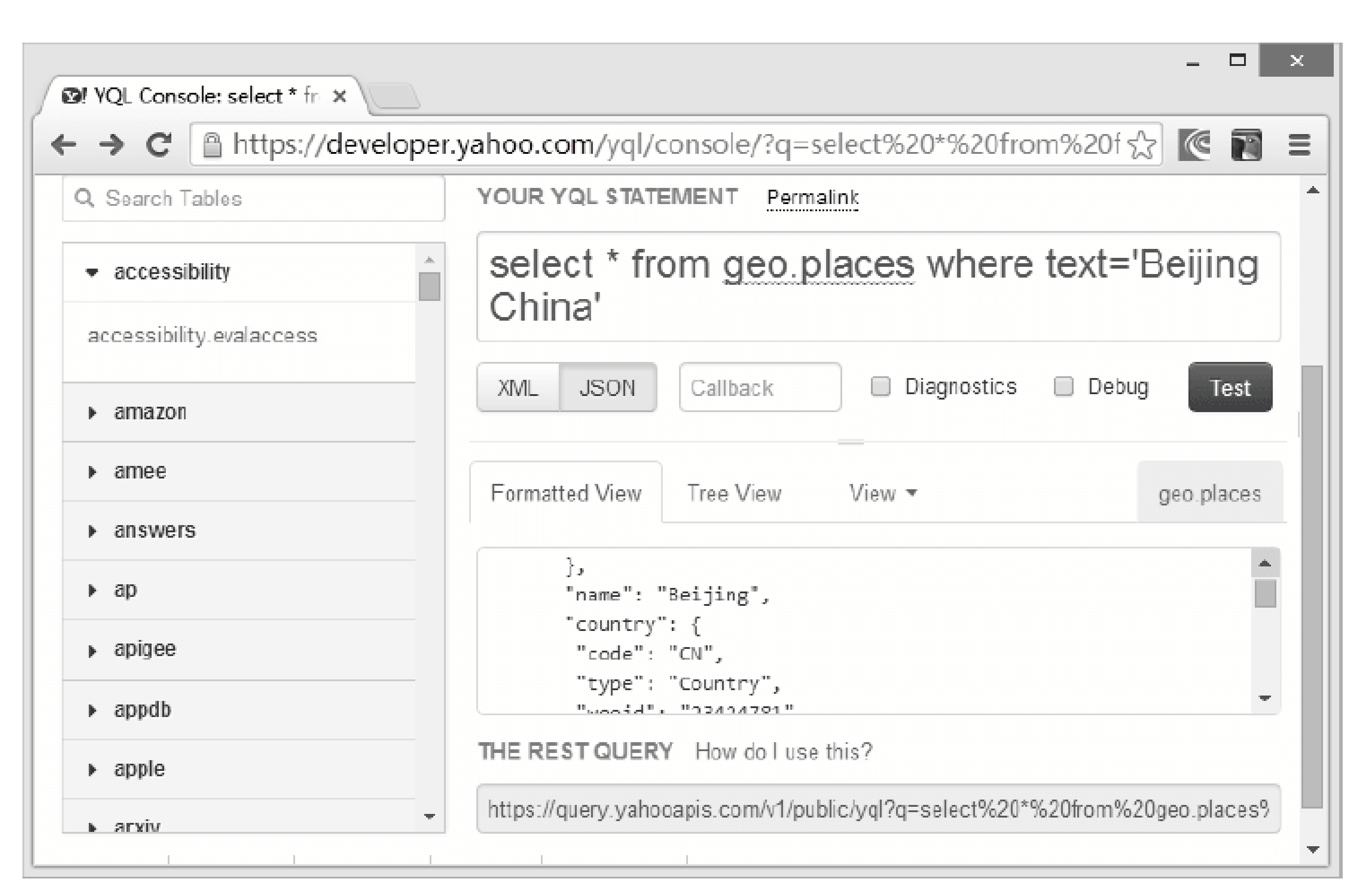


图 12.7　利用 YQL 控制台调试 YQL

此外，还可在控制台的中央的底部看到对应的 REST 查询 URL。例如对于上面的查询语句，其 REST URL 为:

https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20geo.places%20where%20text%3D'Beijing%20China'&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys&callback=

也就是说 REST URL 是由 http://query.yahooapis.com/v1/public/yql?加查询语句组成的。

有了该 REST URL，我们便可在应用程序中利用 ArcGIS API for JavaScipt 的 esri/request，或 Dojo 的 dojo/request/script.get，或 jQuery 的$. ajax 来调用，即可得到如同在“Formatted View”中显示的查询结果。

在 YQL 控制台页面的最右部分列出了 Yahoo!提供的所有可查询的表。当然我们也可以输入“show tables”查询语句来获取该列表。当鼠标移动到某个表时，会出现一名为“desc”的按钮，单击该按钮，便可查看该表的描述。

由于 Yahoo!提供了多方法的查询表，通过组合查询，可得到一些复杂的结果。例如要在页面中显示用户当前所在城市的天气预报，可按下面的方式来实现:

（1）利用 HTML5 中的 geolocation 来得到设备的地理位置；

（2）构造一个 YQL 语句来查询 geo.placefinder 表，得到该地理位置所在城市的 woeid（Where On Earth Identifier）；

（3）利用该 woeid，构造一个查询语句，查询 weather.forecast，便可得到今后 5 天的天气预报。

12.5　实践 19：从 OpenStreetMap 获取源数据

从 OpenStreetMap 中获取数据要比往里面添加数据更困难。当往 OSM 中增加数据时，可以使用许多不同的编辑器，也可以使用任何标签，当然需要尽量遵守标签规范。

当从 OSM 中获取数据时，需要考虑如下问题：

（1）只获取需要的标签数据；

（2）获取需要的数据格式；

（3）不要获取过太多的数据，以免服务器与自己难以承担。

此外，另一个突出的问题是从 OSM 返回的 XML 数据使用的是它自己的结构，许多 GIS 应用程序不能直接读取。因此还需要将该 XML 转换为其他格式。

有多种机制下载 OSM 数据。应对上述这些挑战最简单的机制是，提供一种方法来过滤想要的标签，允许指定输出格式，并允许指定限制请求数据的一个地理边界框，这样就不会取得过多数据。

也有许多方式来下载 OSM 数据。其中一个用户最友好的具有图形界面的方式是通过 BBBike.org 服务器，网站地址是 http://extract.bbbike.org，如图 12.8 所示。通过该基于 Web 的工具，可以交互式地绘制一矩形框，并指定希望的输出格式。稍过一会儿，该网站会将一个链接地址通过邮件的方式发送给你，通过该链接便可下载数据。

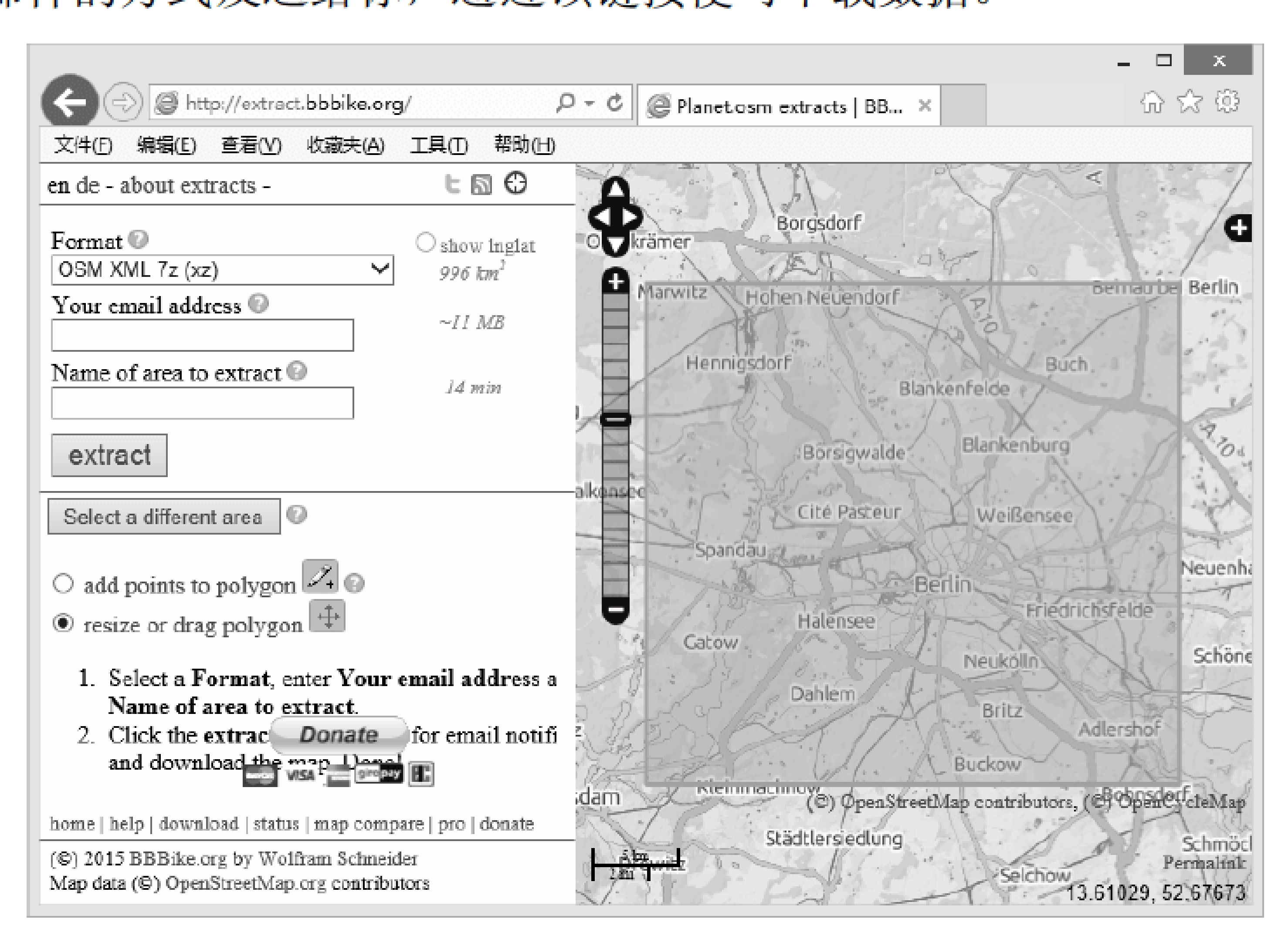


图 12.8　通过 BBBike.org 服务器下载 OSM 数据

在本实践的第一部分中，我们将介绍如何使用 QGIS 来下载 OSM 数据，该方式比通过 BBBike.org 服务器更直接，而且对选择标签具有更大的灵活性。在第二部分将介绍如何使用 XAPI 来下载数据。

12.5.1 使用 QGIS 下载数据

本实践以从 OpenStreetMap 中下载德国波恩城市的建筑物为例，来演示如何使用 QGIS 下载数据。

（1）下载数据。

启动 QGIS，选择“矢量”菜单中的“开放街道图”子菜单中的“Download Data”命令，打开图 12.9 所示的“下载开放街道图数据”对话框。在该对话框中，手动输入要下载数据范围的经纬度。然后设置输出文件的路径与文件名。最后单击“确定”按钮，下载数据。



图 12.9　通过下载开放街道数据对话框下载数据

（2）从.osm 文件中导入数据。

选择“矢量”菜单中的“开放街道图”子菜单中的“Import Topology from XML”命令，打开“导入开放街道图”对话框，并按照图 12.10 填写对话框。最后单击“确定”按钮，将数据导入到一个 SpatiaLite 数据库中。接下来还需要从该数据库中提取指定标签的几何图形，创建有用的图层。



图 12.10　导入开放街道图对话框

（3）输出图层。

选择“矢量”菜单中的“开放街道图”子菜单中的“Export Topology to SpatiaLite”命令，打开“将开放街道图拓扑结构导出到 SpatiaLite”对话框。输入数据库文件设置为第二步创建的 bonn.osm.db，导出类型为多边形，导出标签包含 building、addr:street、addr:housenumber、source 以及 amenity 这 5 个，如图 12.11 所示。最后单击“确定”按钮。



图 12.11　导出图层

（4）查看数据。

通过“将开放街道图拓扑结构导出到 SpatiaLite”对话框，将指定标签数据导出完成后，将在 QGIS 地图中显示 bonn_polygons 图层，其中包含了数据库中的所有多边形，如图 12.12 所示。

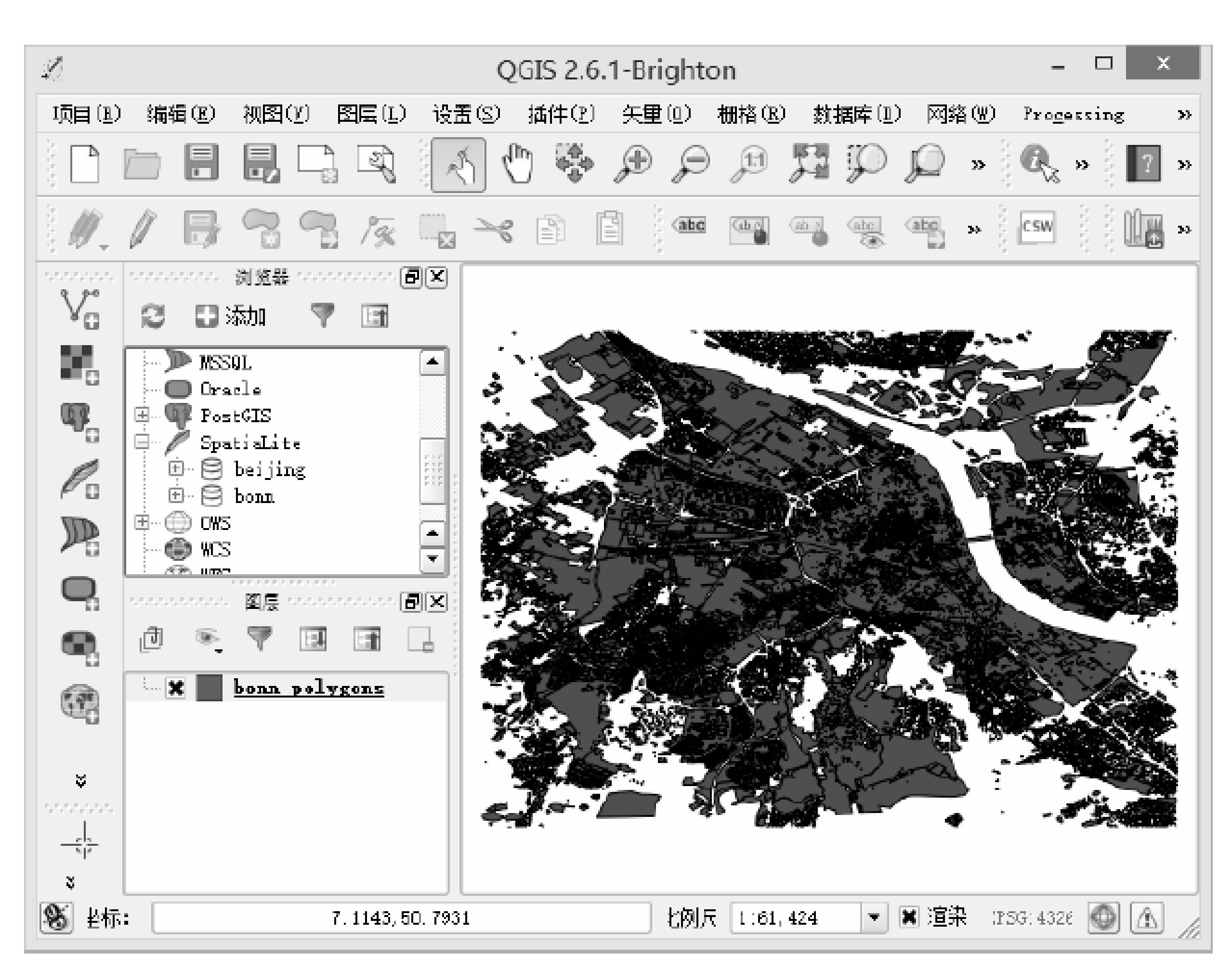


图 12.12　导出的多边形图层

（5）通过表达式选择要素。

在图层管理器中，用鼠标右键单击 bonn_polygons 图层，选择其右键菜单的“Open Attribute Table”命令，打开属性表对话框。在该对话框的顶部选择“使用表达式选择要素”按钮，打开“Select by expression-bonn-palygons”对话框，如图 12.13 所示。在该对话框的“表达式”框中，输入如下查询条件：

```
"building" != 'NULL' AND "building" != 'no'
```



图 12.13　使用表达式选择要素

该表达式过滤了不是建筑物的要素。如果感兴趣的是其他类型的要素，则需要创建只选择需要标签的表达式。

然后单击“选择”按钮。这时在地图中可以看到所有建筑物都被选择了。

（6）将选择要素另存为图层。

在图层管理器中，用鼠标右键单击 bonn_polygons 图层，选择其右键菜单中的“另存为”命令，打开“矢量图层另存为”对话框。在该对话框中输入另存文件的路径与文件名，并选中“Save only selected features”复选框，如图 12.14 所示。最后单击“确定”按钮。



图 12.14　将选中的要素另存为矢量文件

（7）核实输出图层确实只包含建筑物。

另存完成后，QGIS 也会将该文件添加到地图中。通过图层管理器，关闭 bonn_polygons 图层，只显示 bonn_buildings 图层，结果如图 12.15 所示，这便是我们希望的内容与格式数据。

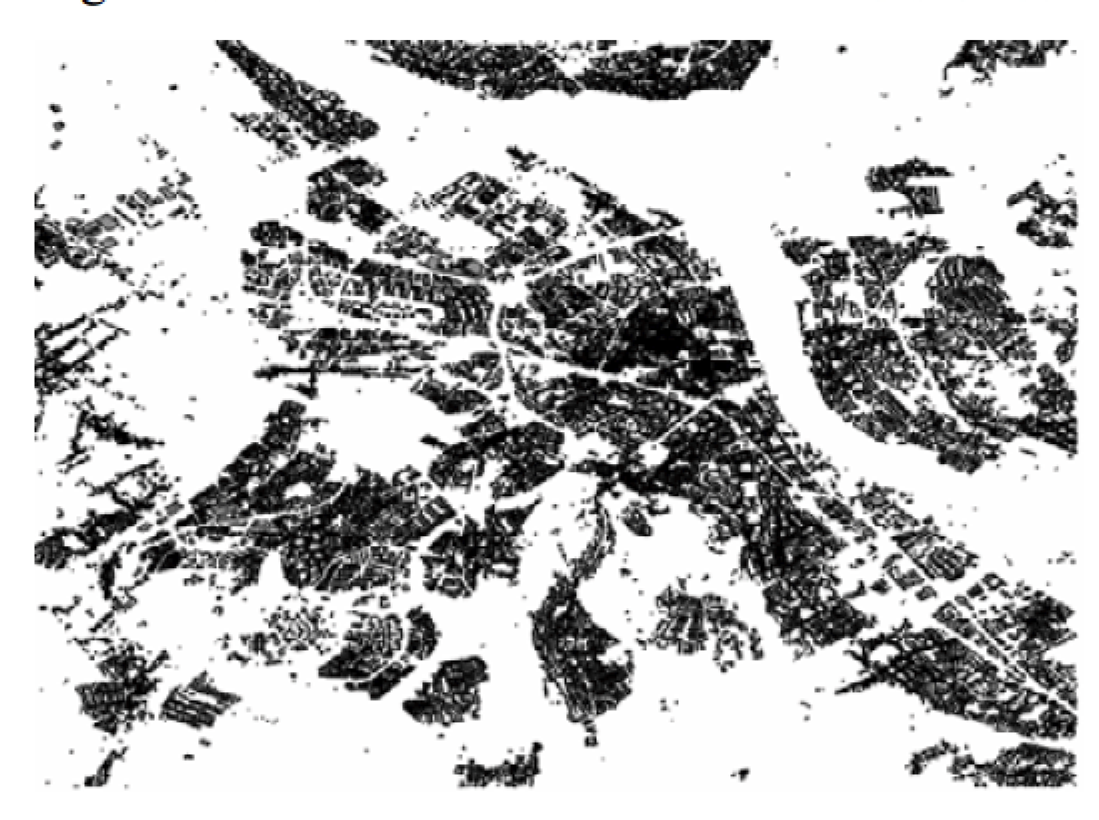


图 12.15　只包含建筑物要素的图层

12.5.2　使用 OpenStreetMap 查询 API 下载数据

OSM 提供了一个主要的 API 用于编辑数据库中的内容。但是大部分使用 OSM 数据库的应用程序却并不用于编辑数据，而只是显示数据库中的内容。为了实现显示要求，需要使用不同的选择条件执行查询，这通常会返回大量的地理数据。因此为了满足这些以显示为目的的需求，OSM 以一种效率更高的方式提供了几个只读接口。

最初开发的是基于主要 API 的名为 XAPI（eXtended API）的只读 API，可实现增强的搜索功能。

后来，开发了使用更强大查询语言的 Overpass API。Overpass API 接受两种查询语言，分别是 Overpass QL 与 Overpass XML，前者是后者的一个简化版本。为了兼容，Overpass API 中也支持使用 XAPI 来查询。当前提供 Overpass API 服务的主要有 3 个著名的服务器，它们分别是：

- http://overpass.osm.rambler.ru
- http://www.overpass-api.de
- http://api.openstreetmap.fr

此外，还有一个名为 Overpass turbo 的基于 Web 的 OSM 数据挖掘工具，其访问地址是 http://overpass-turbo.eu。该工具可运行任何 Overpass API 查询，并以交互地图与数据两种方式显示查询结果。

（1）使用 XAPI 获取数据。

在浏览器地址栏中输入如下地址：

http://www.overpass-api.de/api/xapi_meta?*[building=yes][bbox=7.01,50.68,7.15,50.78]

当响应返回时，会提示保存文件，将文件保存为 buildings.osm。在文本编辑工具中打开该文件，就会看到使用 OSM 格式化的 XML 表达的波恩城市的建筑物数据。

从网址中包含 xapi，可看出使用的是 XAPI 来查询 OSM。

（2）使用 Overpass API 获取 JSON 格式的数据。

在浏览器地址栏中输入如下地址：

http://overpass.osm.rambler.ru/cgi/interpreter?data=[out:json];node[name~holtorf](50.7,7.1,50.78,7.20);out;

该请求用于查询德国波恩城市节点名称中包含“holtorf”的要素。响应结果是图 12.16 中所示的包含 16 个节点的 JSON 格式的数据。

{
  "version": 0.6,
  "generator": "Overpass API",
  "osm3s": {
    "timestamp_osm_base": "2015-04-19T13:13:02Z",
    "copyright": "The data included in this document is from www.openstreetmap.org. The data is made available under ODbL."
  },
  "elements": [

{
  "type": "node",
  "id": 69291491,
  "lat": 50.7330071,
  "lon": 7.1841348,
  "tags": {
    "VRS:gemeinde": "BONN",
    "VRS:ortsteil": "Holtorf",
    "VRS:ref": "68676",
    "bus": "yes",
    "name": "Niederholtorf Mitte",
    "public_transport": "stop_position"
  }
},

图 12.16 请求指定范围内满足条件的 JSON 格式的数据

从请求地址可以看出，Overpass API 的语法与 XAPI 完全不同。更重要的是，指定范围时两者经纬度的顺序也完全相反。

（3）使用 Overpass API 获取 XML 格式的数据。

Overpass API 默认返回的就是 XML 格式的数据，因此在前面的地址中去掉“[out:json];”，便可返回 XML 格式的数据。请求格式如下：

http://overpass.osm.rambler.ru/cgi/interpreter?data=node[name~holtorf]%2850.7,7.1,50.78,7.20%29;out;

（4）使用 OpenLayer 查看查询结果。

Overpass API 服务器还提供了直接使用 OpenLayers 查看返回数据的功能。例如对于上述请求，使用 Overpass API 查询 OSM，并使用 OpenLayers 查看查询结果的地址如下：

http://overpass-api.de/api/convert?data=node[name~holtorf](50.7,7.1,50.78,7.20);out;&target=openlayers&zoom=12&lat=50.72&lon=7.1

该请求返回图 12.17 所示的响应。

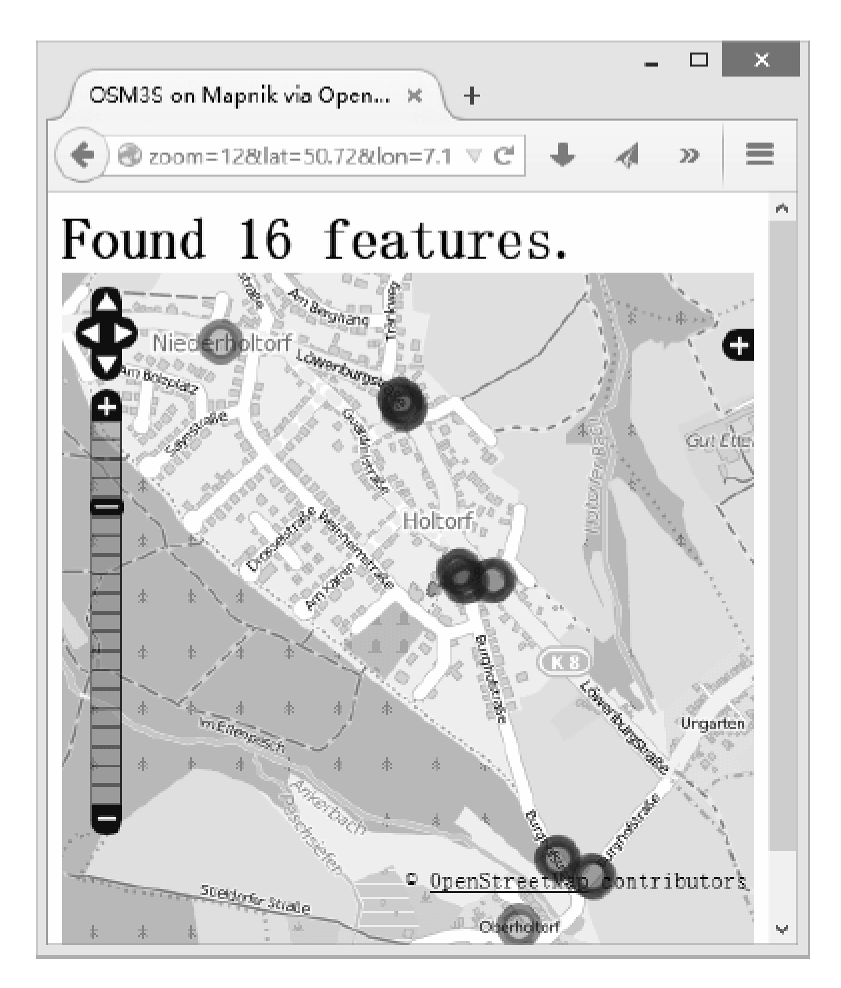


图 12.17　使用 OpenLayers 查看查询结果

（5）使用 Overpass turbo 查询并下载数据。

在浏览器中，进入 http://overpass-turbo.eu 页面。在查询文本框中输入如下内容：

```
[out:json];
way["building"="yes"](50.71,7.07,50.72,7.08);
(._;>;);
out;
```

上述查询用于获取德国波恩小范围地区的建筑物。由于在浏览器中运行，如果将范围设置得很大，所返回的数据会很多，浏览器将难以承受。因此，我们这里使用了比较小的地理范围。

然后单击“Run”按钮，执行查询。查询完成后，在地图窗口的工具栏中单击“zoom to data”按钮，定位到查询获取的数据位置。便可看到查询得到图 12.18 所示的结果。

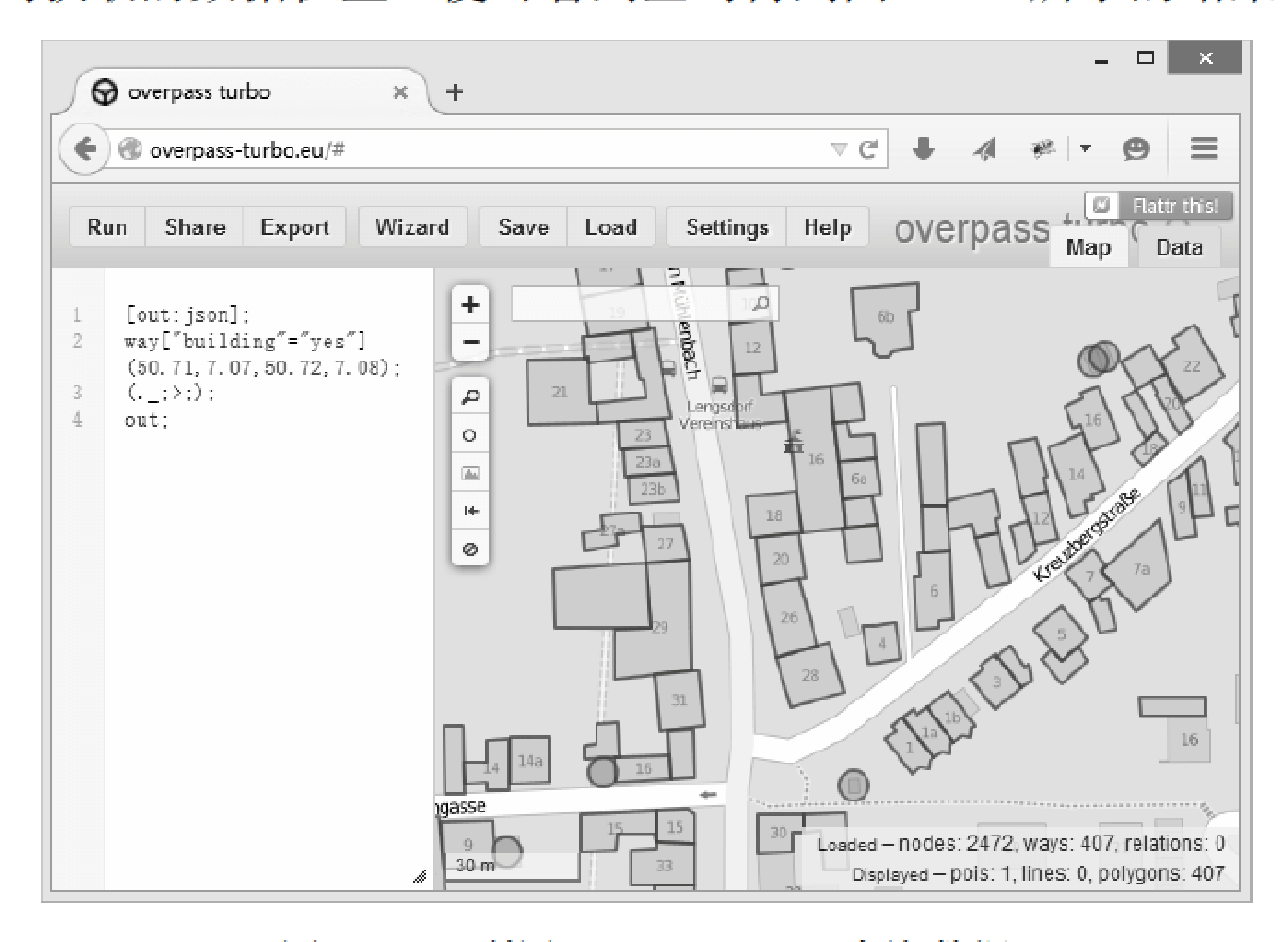


图 12.18　利用 Overpass turbo 查询数据

在地图窗口的右上面，可单击“Data”按钮来查看数据。

得到查询的数据后，便可单击“Export”按钮来输出指定格式的数据。

12.6　实践 20：城市天气预报系统开发

本实践以实例来说明如何融合 Yahoo!的服务，以实现全世界主要城市天气预报。

12.6.1　服务准备与页面设计

（1）发布服务。

将下载文件“Data\Major_World_Cities”文件夹中的 Major_World_Cities.shp 文件发布为服务。该数据包含了世界主要城市的点数据，空间位置字段为 the_geom，城市名称字段为 NAME。

（2）页面设计。

新建一个名为 Walkthrough21 的文件夹，在其中加入名为 WeatherForecast.html 的文件。该文件的内容如下：

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="utf-8">
	<title>混搭 Yahoo!的天气服务</title>
	<link href="css/Main.css" rel="stylesheet" />
	<script src="http://cdnjs.cloudflare.com/ajax/libs/openlayers/2.13.1/OpenLayers.js">
	</script>
	<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
	<script src="WeatherForecast.js"></script>
</head>
<body onload="init()">
	<div id="map"></div>
</body>
</html>
```

页面将样式都放置在 css 文件夹中的 Main.css 文件中了，JavaScript 代码都放置在 WeatherForecast.js 文件中。

12.6.2　代码实现

（1）设置全局变量。

在 Walkthrough21 文件夹中新建一个名为 WeatherForecast.js 文件。在其中加入如下代码，用于设置全局变量：

```
var WGS84 = new OpenLayers.Projection("EPSG:4326"); // WGS 1984
var mercator = new OpenLayers.Projection("EPSG:900913"); // Web 墨卡托投影
var map , selectControl;
var weatherIconMap = [
    'storm', 'storm', 'storm', 'lightning', 'lightning', 'snow', 'hail', 'hail',
    'drizzle', 'drizzle', 'rain', 'rain', 'rain', 'snow', 'snow', 'snow', 'snow',
    'hail', 'hail', 'fog', 'fog', 'fog', 'fog', 'wind', 'wind', 'snowflake',
    'cloud', 'cloud_moon', 'cloud_sun', 'cloud_moon', 'cloud_sun', 'moon', 'sun',
    'moon', 'sun', 'hail', 'sun', 'lightning', 'lightning', 'lightning', 'rain',
    'snowflake', 'snowflake', 'snowflake', 'cloud', 'rain', 'snow', 'lightning'
];
```

（2）地图初始化并加入 WFS 图层。

在上述代码下面，加入如下代码，用于实现地图初始化，并在地图中加入 OSM 与 Major_World_Cities 专题图层，以及加入监听要素选择事件的控件。

```
function init() {
    map = new OpenLayers.Map("map");

    // 使用 OpenStreetMap 作为基础底图
    var baseLayer = new OpenLayers.Layer.OSM("");

    var wfs  layer = new OpenLayers.Layer.Vector("Major cities", {
        strategies: [new OpenLayers.Strategy.Fixed()],
        protocol: new OpenLayers.Protocol.WFS({
            version: "1.1.0",
            // 数据加载的 URL 路径
            url: "http://localhost:8080/geoserver/wfs",
            // 工作区关联的命名空间  URI
            featureNS: "http://localhost:8080/geoserver/webgis",
            // 图层名称
            featureType: "Major  World  Cities",
            // 空间坐标所在字段名
            geometryName: "the_geom",
            schema:
"http://localhost:8080/geoserver/wfs/DescribeFeatureType?version=1.1.0&;typename=webgis:Major  World  Cities"
        })
    });
```

```
        map.addLayers([baseLayer, wfs_layer]);
        var lonlat = new OpenLayers.LonLat(115.45, 40.3).transform(WGS84, mercator);
        map.setCenter(lonlat, 3);

        selectControl = new OpenLayers.Control.SelectFeature([wfs_layer], {
            onSelect: onFeatureSelect,
            onUnselect: onFeatureUnselect
        });

        map.addControl(selectControl);
        selectControl.activate();
    }
```

（3）获取城市天气预报。

当用户在地图上选择某一个城市时，将调用 onFeatureSelect 函数。我们需要在该函数中实现从 Yahoo!服务获取该城市的天气预报。

在 WeatherForecast.js 文件中，继续加入如下代码：

```
    function onFeatureSelect(feature) {
        var results;
        var q = "select * from geo.places where text='" + feature.attributes.NAME +
            " " + feature.attributes.COUNTRY + "'";
        var yql = 'http://query.yahooapis.com/v1/public/yql?q=' +
            encodeURIComponent(q) + '&format=json';
        var woeid, content;
        $.ajax({
            async: false, url: yql, dataType: 'json',
            success: function (r) {
                if (r.query.count == 1) {
                    woeid = r.query.results.place.woeid;
                }
                else if (r.query.count > 1) {
                    woeid = r.query.results.place[0].woeid;
                }
                q = "select * from weather.forecast where woeid=" + woeid + " and u='c'";
                yql = 'http://query.yahooapis.com/v1/public/yql?q=' +
                    encodeURIComponent(q) + '&format=json';
                $.ajax({
                    async: false, url: yql, dataType: 'json',
                    success: function (r) {
                        if (r.query.results.channel.item.title == 'City not found') {
                            content = '<p>Information unavailable</p>';
```

```
                    }
                    else {
                        var item = r.query.results.channel.item.condition;
                        content  =  '<p>'  +  r.query.results.channel.title  +  '</p><div  id="weather"
class="loaded" ><ul id="scroller">' +
                        '<li><img src="images/icons/' + weatherIconMap[item.code] + '.png"/>' +
                        '<p class="day">当前</p>' +
                        '<p class="cond">' + item.text +
                        '<b>' + item.temp + '°C</b></p></li>';
                        var length = r.query.results.channel.item.forecast.length;
                        for (var i = 0; i < length; i++) {
                            item = r.query.results.channel.item.forecast[i];
                            content +=
                            '<li><img src="images/icons/' + weatherIconMap[item.code] + '.png"/>' +
                            '<p class="day">' + item.day + '</p>' +
                            '<p class="cond">' + item.text +
                            '<b>' + item.low + '°C / ' + item.high + '°C</b></p></li>';
                        }
                        content += '</ul>' +
                        '<button onclick="showPrevSlide()" class="arrow previous"></button>' +
                        '<button onclick="showNextSlide()" class="arrow next"></button>' +
                        '</div>';

                        popup = new OpenLayers.Popup.FramedCloud("chicken",
                            feature.geometry.getBounds().getCenterLonLat(),
                            new OpenLayers.Size(100, 100),
                            content,
                            null, true, onPopupClose);
                        feature.popup = popup;
                        map.addPopup(popup);
                    }
                }
            });
        }
    });
    currentSlide = 0;
}
```

我们将在该函数中首先根据用户选择的城市图形的 NAME 属性来构造 YQL 查询语句，查询 geo.places 数据表，得到该城市的 woeid，然后利用该 woeid 构造一个查询语句，查询 weather.forecast 便可得到天气预报。

在上述代码中利用了 jQuery 的 ajax 的方法执行 REST 查询。

（4）实现分屏显示多天天气预报。

由于 Yahoo!天气服务一般会返回 5 天的天气预报，我们都加入到信息模板的内容中了，但一次只显示一天的天气，对于其他天的天气情况，需要利用前进按钮或后退按钮来查看。前进与后退按钮对于的函数代码如下：

```
function showPrevSlide() {
    showSlide(currentSlide - 1);
}

function showNextSlide() {
    showSlide(currentSlide + 1);
}

function showSlide(i) {
    var weatherDiv = $('#weather');
    var scroller = $('#scroller');
    var items = scroller.find('li');
    if (i >= items.length || i < 0 || scroller.is(':animated')) {
        return false;
    }
    weatherDiv.removeClass('first last');
    if (i == 0) {
        weatherDiv.addClass('first');
    }
    else if (i == items.length - 1) {
        weatherDiv.addClass('last');
    }
    scroller.animate({
        left: (-i * 100) + '%'
        },
        function () {
            currentSlide = i;
    });
}
```

（5）要素取消选择事件处理与信息框关闭事件处理。

加入如下两个函数，分别用于处理要素取消选择事件处理与信息框关闭事件。

```
function onFeatureUnselect(feature) {
    if (feature.popup) {
        map.removePopup(feature.popup);
        feature.popup.destroy();
```

```
            delete feature.popup;
        }
    }

    function onPopupClose(evt) {
        selectControl.unselectAll();
    }
```

（6）样式实现。

样式代码 Main.css 及相关图片，请直接复制下载文件“Codes\Walkthrough21”文件夹中的内容。

（7）代码运行与调试。

在浏览器中打开 WeatherForecast.html 文件，便可查看世界主要城市的天气预报了，如图 12.19 所示。

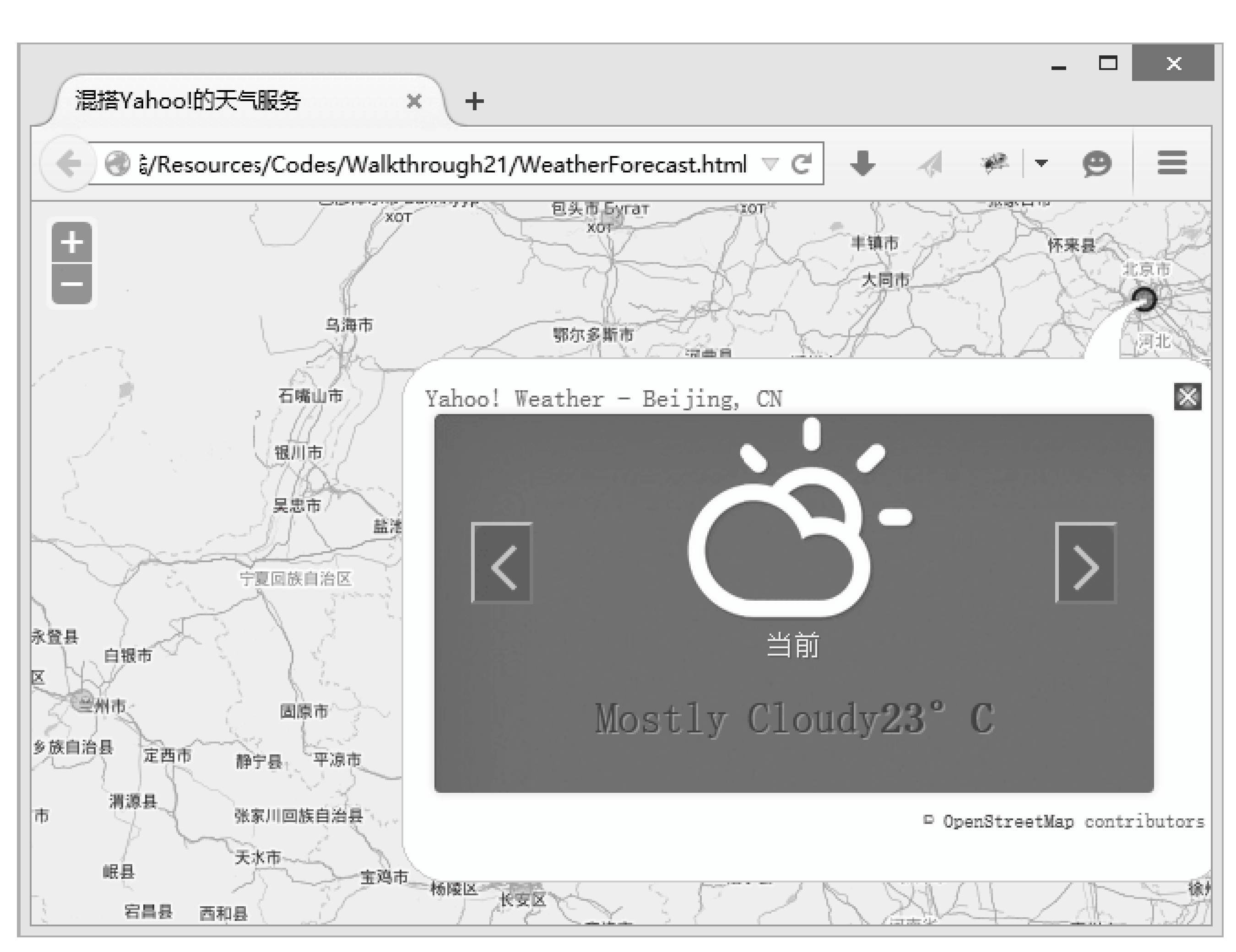


图 12.19　混搭 Yahoo!天气服务

12.7　习　题

（1）仔细阅读以下两个网址的文档，学习 Overpass API 及其语法：

- http://wiki.openstreetmap.org/wiki/Overpass_API
- http://wiki.openstreetmap.org/wiki/Overpass_API/Language_Guide

（2）利用实践 20 介绍的方法，从 OpenStreetMap 下载自己所在城市的街道数据，并将其转为 Shapefile 格式。

（3）开发一个 Web GIS 应用程序，使用 GeoNames 提供的 Web 服务，查询当前地图显示范围内的维基百科文章。参考界面如图 12.20 所示。

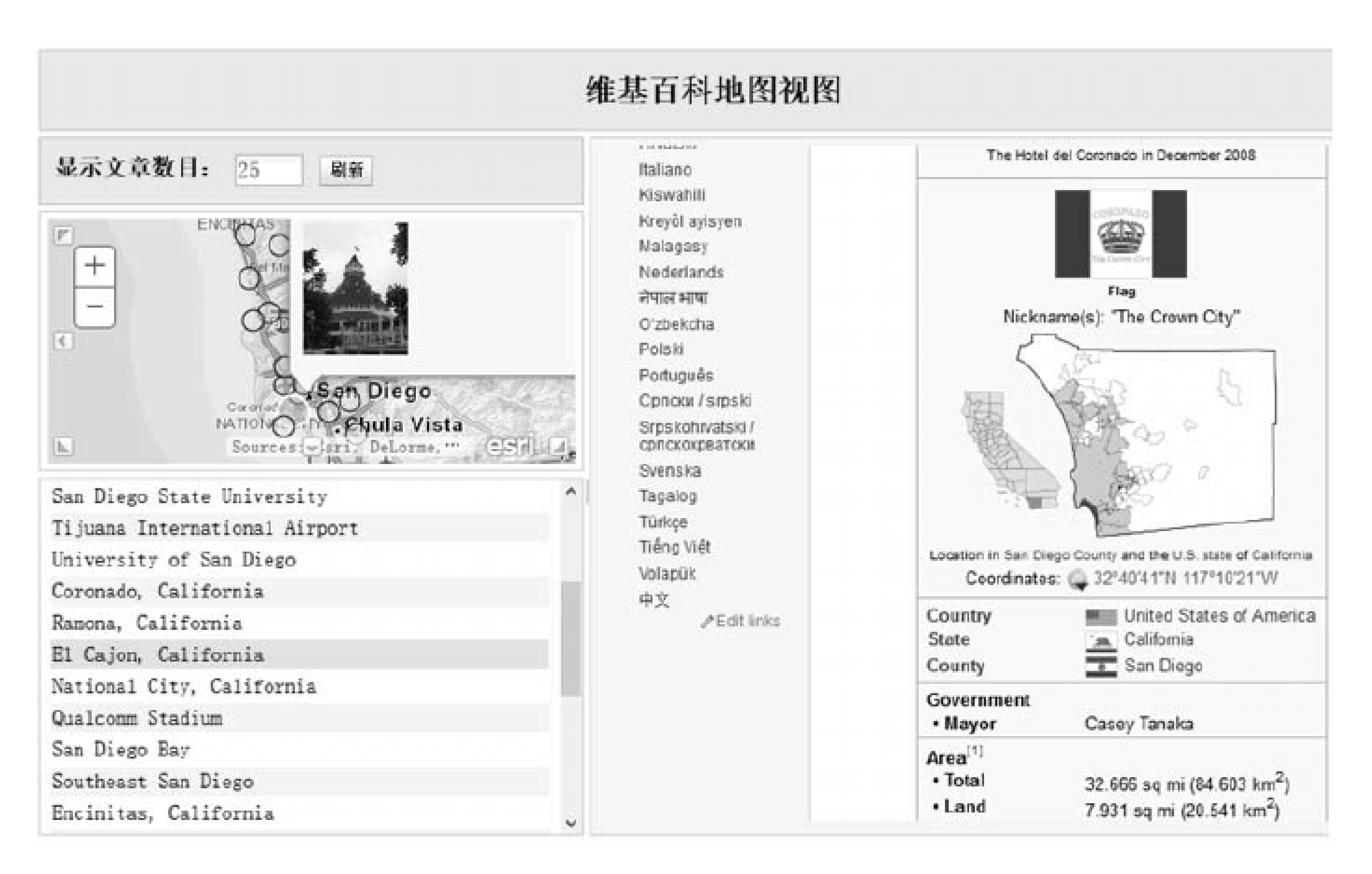


图 12.20　通过地图导航维基百科文章

（4）开发一个 Web GIS 应用程序，应用 Flickr 的相册服务来查询指定标签的所有相册，获取并在地图上显示对应的点要素。单击点要素能显示其对应的相片。参考界面如图 12.21 所示。



图 12.21　混搭 Flickr 的相册服务

（5）在互联网上寻找一个使用 OpenStreetMap 的 Web GIS 应用，编写一个文档，描述如下内容：

- 该应用程序的目的以及 URL；
- 该应用程序是如何使用 OpenStreetMap 的。例如，只是简单地使用了 OpenStreetMap 的地图切片，还是使用了原始数据来创建专题图层等；
- 使用 OpenStreetMap 给该应用程序引入哪些优点和缺点；
- 提出在该应用程序中使用 OpenStreetMap 更恰当的方法。

（6）通过 OpenStreetMap 基于浏览器的编辑器（http://www.openstreetmap.org/），往 OpenStreetMap 数据库中加入自己所在城市或所熟悉的其他地方的数据。